VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Mobilní aplikace pro vizualizaci 3D objektů pomocí rozšířené reality

Mobile Application for Object Visualization Based on Augmented Reality

Vojtěch Maiwald

Bakalářská práce

Vedoucí práce: Mgr. Ing. Michal Krumnikl, Ph.D.

Ostrava, 2021

Abstrakt

Tato bakalářská práce má za cíl prozkoumat možnosti vývoje aplikací v rozšířené realitě, především pro platformu Android. Tento teoretický základ poté bude využit v praxi během tvorby mobilní aplikace včetně visuálního obsahu. Aplikace by měla co nejlépe ukázat a využít potenciál, jaký rozšířená realita na současných telefonech nabízí. Vývoj bude probíhat v jazyce Java s knihovnou ARCore a renderovacím API Sceneform. Pro tvorbu 3D modelů a animací byl zvolen animační software Cinema4D. Po dokončení bude aplikace k dispozici návštěvníkům na hradě Sovinec.

Klíčová slova

Rozšířená realita; ARCore; Sceneform; vývoj pro Android; Java; Cinema4D; animace; hrad Sovinec; bakalářská práce

Abstract

Goal of this bachelor thesis is to explore the possibilities of application development in augmented reality, primarily for the Android platform. This theoretical basis will then be used in practice during the creation of a mobile application, including visual content. The application should show the potential that augmented reality offers on current phones. The development will be in Java language with the ARCore library and the Sceneform rendering API. Cinema4D animation software was chosen to create 3D models and animations. After completion, the application will be available to visitors at Sovinec Castle.

Keywords

Augmented reality; ARCore; Sceneform; Android development; Java; Cinema4D; Animations; Sovinec castle; bachelors thesis

Poděkování

Mé nejupřímnější poděkování patří všem členům mé rodiny za neustálý zdroj podpory a motivace, které se mi dostalo. Mým přátelům, kteří si vždy našli čas na večer s deskovými hrami, kdy jsme si mohli společně odpočinout od školních, ale i jiných povinností. A v neposlední řadě mému skvělému vedoucímu Mgr. Ing. Michalovi Krumniklovi, Ph.D., který mi vždy bez váhání vyšel vstříc, nejen při řešení problémů během psaní této práce, a byl zdrojem velmi cenných a odborných rad všeho druhu. Bez pomoci těchto lidí by bylo velmi obtížné tuto práci dokončit.

Obsah

Seznam použitých zkratek a symbolů

AR	–	Rozšířená realita (Augmented reality)
OS	–	Operační systém (Operating system)
GP	–	Google Play
SDK	–	Softwarová vývojová sada (Software development kit)
NDK	–	Nativní vývojová sada (Native development kit)
ADB	–	Ladící nástroj androidu (Android debug bridge)
FPS	–	Počet snímků za sekundu (Frames per second)
MoCap	–	Zachycení pohybu (Motion capture)
IR	–	Infračervený (Infrared)
JVM	–	Virtuální stroj Javy (Java virtual machine)
ART	–	Běhové prostředí Androidu (Android Runtime)
API	–	Rozhraní pro programování aplikací (Application Programming Interface)
PBR	–	Fyzikální renderování (Physically based rendering)
GUI	–	Grafické uživatelské rozhraní (Graphic User Interface)
UI	–	Uživatelské rozhraní (User Interface)

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Hrad Sovinec bude otevírat novou expozici v nedávno zrekonstruované části objektu a při této příležitosti chce přilákat co nejvíce návštěvníků napříč všemi generacemi. Kromě vystavených info panelů, zbraní a zbrojí bude pro návštěvníky k dispozici i velký model hradu s blízkým okolím. Protože ale upoutat pozornost diváka bývá náročné, rozhodlo se vedení Sovince požádat Fakultu elektrotechniky a informatiky Vysoké školy Báňské o spolupráci na vytvoření mobilní aplikace s interaktivním obsahem v rozšířené realitě. Tato technologie je bezpochyby na vzestupu a po vzoru jiných aplikací doplňujících výstavy muzeí po celém světě o interaktivní obsah bude i tato se snažit zaujmout návštěvníky novým a nevšedním způsobem.

Cílem této práce je prozkoumat, jaké možnosti rozšířená realita nabízí na současných mobilních zařízeních platformy Android a poté tyto poznatky využít při vytváření skutečné aplikace v publikovatelné podobě. Protože rozšířená realita paradoxně na běžných telefonech až tolik rozšířená není, důležité je věnovat se podpoře mezi zařízeními a jejich technickými možnostmi (nejen s ohledem na výpočetní výkon). Pro lepší představu o současném vývoji aplikací pro rozšířenou realitu jsou dostupné technologie i platformy porovnány. Primárně se ale tento text bude zabývat vývojem na platformu Android.

Součástí aplikace pro Sovinec bude divácky atraktivní obsah ve formě animací, které se samy spustí, pokud uživatel nalezne v místnosti příslušný aktivační tag. Záměrem tedy je, aby použití aplikace bylo co možná nejsnazší a zvládl to opravdu každý. Animace budou vizuálně doplňovat informace, které budou v textové podobě rozmístěné různě v rámci výstavy. Aplikace tedy nemá mít vyloženě vzdělávací charakter, ale spíše zábavnou formou doplňovat informace k výstavě, tím ji ozvláštnit a přizpůsobit i mladším generacím, které si v dlouhých textech nelibují.

Podrobně bude v této práci vysvětlen i zvolený postup při tvorbě animací, které jsou důležitou součástí. Je totiž nutné, vzhledem k omezenému výpočetnímu výkonu telefonů oproti počítačům, aby obsah byl jednoduchý na přehrávání. K tomuto jsem použil program Cinema4D hlavně kvůli předchozím zkušenostem, ale také protože to licenční podmínky dovolují.

Teoretický rozbor bakalářské práce je rozdělen do dvou kapitol: **2 (Technologie, knihovny a platformy pro AR)** zabývající se technologiemi rozšířené reality a **3 (Tvorba animací)** zabývající tvorbou a úpravou animací pro snadné použití v rozšířené realitě; protože to jsou z principu odlišné věci, jejich finální propojení je blíže popsáno v kapitole následující: **4 (Praktická část)**. Tam se rovněž nachází postup, jakým byla aplikace postupně tvořena se všemi nástrahami, které bylo nutné odstranit, a problémy, které bylo nutné vyřešit.

Kapitola 2

Technologie, knihovny a platformy pro AR

Tato část práce se bude zabývat teoretickým rozborem vývoje aplikací pro rozšířenou realitu. Budou vysvětleny její základní principy funkčnosti po softwarové stránce, její možné využití v nejrůznějších oblastech, ale i její limity spojené například s hardwarem použitého zařízení.

Jelikož praktická část této práce vznikla s využitím knihovny ARCore na OS Android, je těmto technologiím věnována největší pozornost, vhodné ale je znát i srovnání s jinými dostupnými knihovnami na ostatních existujících platformách.

2.1 Úvod a využití

Augmented reality neboli rozšířená realita, je ozvláštnění reálného světa o objekty, které v něm fyzicky nejsou přítomny, ale byly tam prostřednictvím nějakého zařízení umístěny v reálném čase. K tomu lze využít mobilní telefon, ale v průmyslových odvětvích se spíše setkáme se speciálními brýlemi. Můžeme tak například propagovat své produkty a pomoct lidem doma s vybíráním a umístěním nového nábytku [1]. Na montážních linkách a tréninkových střediscích nejen v automotive průmyslu [2] je tato technologie rovněž pomalu nasazována (viz obrázek **2.1**). Můžeme díky rozšířené realitě ozvláštnit výlet do parku sbíráním Pokémonů [3] nebo zabíjím monster [4] ve stylu Zaklínače. Reálné uplatnění tedy už dnes najdeme od průmyslu, přes výuku až po zábavu díky velice rychlé evoluci technologií a jejich snadnému uplatnění pro veřejnost.

2.2 Operační systém Android

Android je open source operační systém vyvíjený firmou Google od roku 2008 [6]. Je založen na Linuxovém kernelu, které zprostředkovává data z hardwaru do vyšších vrstev systému. Dnes jej na svých zařízeních využívají přední výrobci telefonů kvůli snadné možnosti upravit si základní funkčnosti a vzhled systému pomocí rozšiřující softwarové nadstavby. Existují i skupiny nezávislých vývojářů, kteří na základě originálního zdrojového kódu Androidu tvoří takzvané *custom ROM* -

Obrázek 2.1: Návod na údržbu stroje v AR, převzato z [5]

neoficiální distribuce systému, které často mají za cíl poskytovat nejnovější bezpečnostní záplaty i na starší zařízení, kde je výrobce už nevydává.

Oficiální distributor aplikací na Android je služba Google Play, kde může každý vývojář zveřejnit své aplikace pro veřejnost při dodržení podmínek. Takto lze poskytovat i placený obsah nebo vydělávat na zveřejnění reklam v aplikaci. Google Play sám upozorňuje majitele telefonu na vydání nové verze aplikace nebo v případě jiného nastavení se na pozadí sám postará o aktuálnost aplikací instalací nejnovějších verzí.

2.2.1 Vývoj aplikací pro Android platformu

Aplikace lze vyvíjet mnoha velmi odlišnými způsoby a každý má své výhody i nevýhody [7]. Vývojář by si měl nejprve sám ujasnit, co od aplikace požaduje, protože tím si lze práci velmi zjednodušit nebo naopak velmi ztížit. Pokud je požadována multiplatformnost, tak není nutné paralelně vytvářet více verzí stejné aplikace na různé platformy, ale stačí vhodně zvolit framework, který toto umožňuje.

2.2.1.1 Java, Kotlin

Oba jazyky ke svému chodu potřebují JVM, který je za běhu aplikace překládá z Java bytekódu do nativně spustitelné podoby. [8] Java byla dlouho jediný jazyk pro vývoj na Android, ale s rozvojem tohoto OS se postupně přidávaly další možnosti. Kotlin se v současné době považuje za preferovanější alternativu než samotná Java, protože byl vytvořen mnohem později a mohl tak reflektovat

nejčastější problémy Javy, jako třeba často kritizovaná dlouhá syntaxe. Přestože pro vývoj na Android bude Java jistě ještě dlouho dostupná, již nyní Google upřednostňuje Kotlin a má pro něj vytvořenou kompletní dokumentaci. Jeden z důvodů opouštění od Javy je i táhlý spor Googlu se společností Oracle [9].

2.2.1.2 C, C++

Díky balíku NDK je vývoj na Android možný i rovnou v nativní podobě za použití jazyků C nebo C++ [7] , které se překládají do strojového kódu. Ten je sice spouštěn na zařízení pomocí JVM, ale překlad již byl proveden během sestavování aplikace. Takto lze nejefektivněji využít výpočetní sílu daného telefonu, pokud ji aplikace vyžaduje.

2.2.1.3 Go, Dart

Alternativní nativní vývoj lze realizovat v jazycích Go [10] nebo Dart [11], oba jsou vyvíjeny Googlem a umožňují moderní objektové přístupy při zachování efektivity podobné C++. Jako jeho náhradu je ale považovat nemůžeme, kvůli méně obecnému spektru využití. Obsahují bezpečný přístup pro práci s pamětí, garbage collector a jsou kompilovatelné do JavaScriptu - tedy vhodné i pro vývoj webu.

2.2.1.4 Xamarin - C#

Multiplatformní nativní vývoj je možný díky frameworku Xamarin, založeném na technologii .NET [12]. Cílem bylo spojit pohodlnost psaní kódu v jazyce C#, rychlost nativně spouštěného kódu a multiplatformnost. To se sice podařilo částečně splnit, ale za cenu mnoha kompromisů. Pokročilejší možnosti jazyka C# jsou velmi omezeny, což přináší i dosti problematickou rozšiřitelnost pomocí *NuGet* balíčků třetích stran [13]. Podpora tvorby UI je rovněž multiplatformností velmi poznamenaná a často se vyvíjí na každou platformu zcela zvlášť.

2.2.1.5 Flutter - Dart

Flutter je nástroj pro tvorbu aplikací a webů v jazyce Dart, založený na tvorbě a vykreslování *widgetů*, což mohou být jakékoliv zobrazitelné elementy. Během vývoje se kód kompiluje metodou *just in time* což umožňuje tzv. *hot reload*, tedy nasazení změn do aplikace za jejího běhu bez ztráty stavu za pomoci Dart virtual machine spuštěného v prostředí Flutter Desktop Embedding. Flutter ve verzi 2.0 [14] (vydaný 3. března 2021) již má vlastní běhové prostředí kompatibilní s desktopovými OS. Vývoj mobilních aplikací je multiplatformní a ve výchozím nastavení pracuje s *Material designem*, který je dnes velmi populární.

2.2.1.6 React Native - React

React Native je technologie firmy Facebook, která umožňuje spojení tvorby interaktivních komponent pomocí jazyka JavaScript a nativního volání systémových funkcí na zařízení [15]. React umožňuje použití JSX, pro nahrazení HTML kódu, a standardu ES6, pro kratší a srozumitelnější syntaxi. Stejně jako v samotném Reactu se obsah vytváří pomocí funkcionálních nebo objektových komponent a jejich stavu.

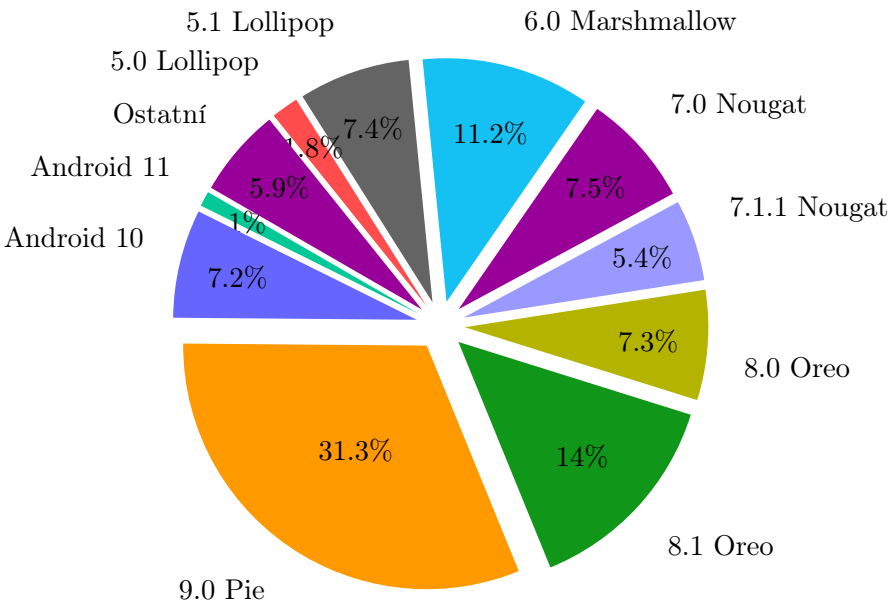
2.2.1.7 Web - JavaScript + HMTL + CSS

Takto vytvářené aplikace se označují jako hybridní, protože obsahují společnou část v podobě webové stránky a ta se poté spojí s nativní částí různou pro každou platformu. Práce s nativní částí se ale napříč platformami liší a vznikají tak duplicitní úseky kódu. Utvářet aplikace tímto způsobem lze např. přes framework Apache Cordova nebo Ionic.

2.2.2 Android Studio

Multiplatformní editor kódu postavený na IntelliJ pro psaní zdrojových kódů primárně Android aplikací v jazycích Java, Kotlin, C++, ale i dalších [16]. O jeho vývoj se přímo stará Google a lze jej použít nejen pro tvorbu aplikací pro telefon, ale i na chytré hodinky s Wear OS, Android TV nebo platformu Android Auto. Pro všechny tyto typy projektů jsou již vytvořeny základní projekty, ze kterých lze vycházet, což je ideální volba pro začínající vývojáře, kteří si chtějí vše od základů vyzkoušet. Při vytváření nového projektu musíme zvolit minimální verzi Android API, což představuje nejnižší verzi systému, na kterém aplikace bude spustitelná. Takto je zajištěna podpora nových funkcí, API, knihoven a možností programovacího jazyka. Vždy, když u nového projektu konkrétní Android API zvolíme, ukáže nám Android Studio na kolika procentech reálných zařízení bude aplikace podporována. Z toho jsem jednoduše vypočítal procentuální zastoupení jednotlivých verzí a výstupní graf je vidět na obrázku **2.2**.

Přímo do vývojového prostředí jsou implementovány podpůrné doplňky, které mají za úkol co nejvíce zautomatizovat a zjednodušit. Některé jsou od základu součástí Android Studia a mnoho dalších jsou dílem komunity vývojářů a lze je formou pluginů doinstalovat. Důležitý je *Gradle*, což je nástroj, který se stará o sestavování aplikací [17], jeho konkrétní nastavení pro daný projekt můžeme najít v souboru *build.gradle*. Užitečná je rovněž možnost propojení lokálního projektu s verzovacím systémem (např. GitHub) a tím mít možnost správy *branchí*, *commitů* a v případě týmu mnoha lidí pracujících na stejném projektu i *konfliktů* a *mergování*. Další doplňky slouží např. pro správu nainstalovaných Android SDK balíčků, úpravy a vytváření emulátorů nebo profilování chování hardwaru zařízení během ladění aplikace. Ladění lze provádět přímo na reálném zařízení přes ADB [18] připojené kabelem k počítači nebo přes WiFi s odchytáváním systémových výpisů nebo výjimek. Stejného lze docílit vytvořením Android emulátoru spustitelného přímo přes Android Stu-



Obrázek 2.2: Procentuální zastoupení verzí Androidu mezi koncovými uživateli k lednu 2021

dio. Jedním klikem lze napsanou aplikaci sestavit, nainstalovat na reálném či emulovaném zařízení a spustit, případně rovnou za chodu aplikace nahrát změny.

2.2.3 Android emulátor

Tímto nástrojem si lze vytvořit téměř neomezenou sbírku virtuálních telefonů na svém počítači. Lze si nastavit nejen verzi Android OS na aktuální nebo historické verze, ale i hardware podle vzoru skutečného zařízení nebo si vytvořit vlastní. Nejčastější volbou jsou ale již přednastavené telefony a tablety řady Pixel vydávané přímo Googlem, které jako jediné mohou obsahovat i GP.

Emulátor je schopen i emulovat data z různých senzorů a modulů v telefonu, jako například akcelerátor, GPS, fotoaparát, snímač otisků prstů, nabíjení baterie, poloha ze signálu (WiFi, 4G), ale nabízí i možnost simulace hardwarových tlačítek. Pro většinu možných potřeb emulátor lze považovat za plnohodnotnou náhradu skutečných telefonů.

S vývojem knihovny ARCore vznikla potřeba takovéto aplikace nějak efektivně ladit. Protože ale ke správné funkčnosti rozšířené reality je třeba s telefonem hýbat, aby si dobře zmapoval okolní prostředí, ladění kabelem přes ADB [18] není tolik praktické a navíc ne všichni k tomu mají v okolí počítače vhodné prostředí. Zřejmě proto vznikl VirtualScene [20], viz obrázek **2.3**. Jde o virtuální byt, ve kterém se lze snadno pohybovat a otáčet všemi směry a tím adekvátně ovlivňovat data z emulovaného akcelerometru. V bytě se nachází i modifikovatelné plochy a to konkrétně jeden obraz pověšený na stěně a plocha na stole. Na tyto plochy lze umístit libovolný obrázek z počítače, a tak testovat například detekci AR tagů.

Obrázek 2.3: AR model lišky umístěný ve VirtualScene, převzato z [19]

2.3 Dostupné knihovny pro AR

Dostupných knihoven je velké množství, některé se zaměřují velice úzce, jiné mají snahu být co nejobecněji použitelné i co se týká platforem. Vždy ale platí, že knihovny pro práci s rozšířenou realitou se starají jen o práci s okolním prostředím, jeho analýzou a pohybem zařízení v prostoru. Není v nich nijak řešený rendering objektů, animací, obrázků nebo videí, to vše se musí obstarat použitím externího API. Na Android telefonech to může být OpenGL nebo Filament, případně vysokoúrovňové API Sceneform (založené na Filamentu). Na zařízeních s iOS a macOS lze využít nízkoúrovňový Metal nebo z něj vycházející výše postavené řešení RealityKit, SceneKit nebo SpriteKit. Herní enginy využívají vlastní řešení, která často vycházejí z těch již dostupných na danou platformu. Použité API je důležité volit i podle toho, jak moc jsme zběhlí programátoři, vysokoúrovňová řešení bývají velmi přívětivá pro svou krátkou a srozumitelnou syntaxi, ale složitější úkoly může být nutné řešit na nižších úrovních. Hlavně pokud je klíčová efektivita práce s malým výpočetním výkonem.

2.3.1 ARCore

Této knihovně je věnována samostatná podkapitola **2.4 (ARCore)**.

2.3.2 ARKit

ARKit je Applem vyvíjená knihovna určená exkluzivně pro operační systém iOS [21]. Tím, že Apple zařízení je výrazně méně druhů a s velmi podobným hardwarem, může být vývoj knihovny veden jen tímto směrem a u žádného zařízení pak není třeba dělat kompromisy. Což s sebou přináší i

jistotu pro vývojáře, chování bude stejné napříč zařízeními. Výhoda ekosystému Apple je zároveň i nevýhodou, na jiných OS tuto knihovnu nelze použít. Zdrojový kód není veřejně dostupný, ale za to je vývojářům dostupná velmi podrobná dokumentace s dostatkem ukázkových kódů. Pro psaní aplikací lze použít jazyk Swift, ale i Objective-C.

Kromě běžných funkcí společných pro většinu zdarma dostupných knihoven nabízí ARKit i velmi zajímavou možnost práce s reálnými 3D objekty, která je jinak k dispozici až u komerčních řešení [22]. Reálné 3D objekty lze skenovat s exportem do .usdz formátu pro pozdější editaci nebo lze 3D model využít jako kotevní bod, což v praxi znamená, že je nutné ho nejprve v prostoru najít a úspěšně rozpoznat, a to za různých světelných podmínek a ze všech úhlů pohledu, tím se velmi okrajově zabývá tato práce [23]. Autor dokonce sám velmi chválí, jak je funkce plynulá a dobře odladěná na jeho zařízení.

Veškerá práce s 3D modely za běhu aplikace probíhá ve formátu .usdz, u jinde vytvořených modelů bude tedy nutná konverze. Ta lze zařídit např. programem Reality Composer, který je dostupný na zařízeních s macOS a je plně vybaven na práci s modely pro AR v knihovně ARKit.

2.3.3 AR Foundation

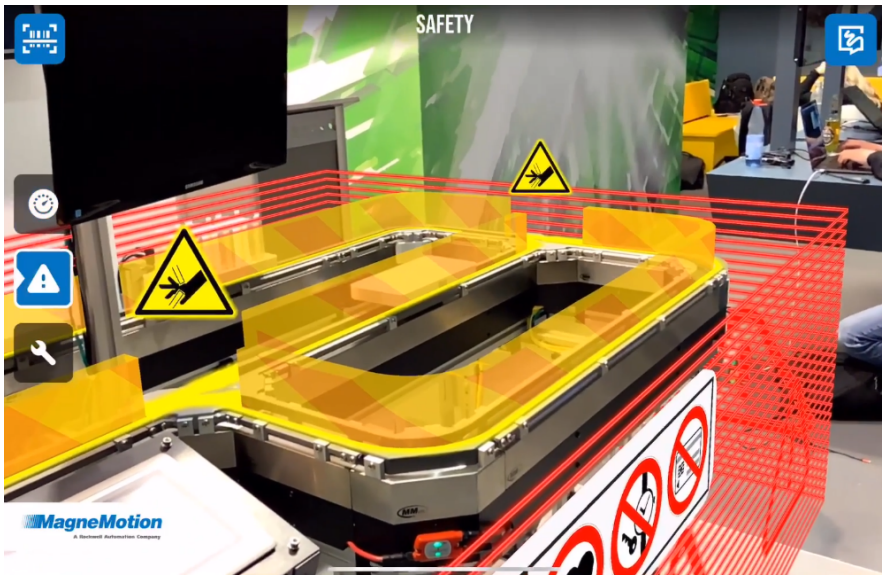
AR Foundation je knihovna se kterou se pracuje v případě použití Unity enginu [24]. Sama žádné pokročilé funkce neimplementuje, ale pouze dovoluje vytvářet jeden projekt pro více platforem a během sestavování aplikace se podle zvolené platformy interně volá ARCore XR Plugin pro Android, ARKit XR Plugin pro iOS nebo některá z dalších knihoven v případě jiné platformy. Renderování ale probíhá vždy v rámci Unity enginu.

Pro tvorbu interaktivního obsahu do aplikace úplně stačí využít možnosti, které samo Unity nabízí ve svém prostředí a není tak potřeba používat jiný software. Jedná se o herní engine využívaný i velikými herními studii, což vypovídá o velikém potenciálu a kvalitě zpracování. Pokud jde o funkční stránku tvorby v Unity, tak to podporuje programování v jazyce C# ve frameworku Mono, což je menší, a hlavně multiplatformní verze .NET frameworku. Editor je kromě Windows a macOS od nedávna dostupný i na Linuxové OS.

2.3.4 Vuforia

Vuforia je zpoplatněná, ale velmi profesionálně vytvořená platforma pro práci s AR s cílovým uplatněním hlavně v průmyslu [25], což je ukázáno na obrázku **2.4**. Součástí řešení je kompletní podpora Androidu, iOS i Universal Windows Platform. Vuforia pracuje s vlastním renderovacím API s názvem Vuforia engine, který lze použít i v prostředí Unity. Pro tvorbu modelů je doporučována práce ve Vuforia Studio, které je uzpůsobené na import a práci s modely vytvořenými ve standardu CAD.

Práci s touto knihovnou si lze vyzkoušet zdarma, nicméně výslednou aplikaci nelze publikovat. Nejlevnější licence s mírně omezenou funkčností stojí 504$ na rok provozu [26]. Vuforia nabízí



Obrázek 2.4: Pohled na výrobní linku v AR aplikaci vytvořenou pomocí knihovny Vuforia, převzato z [27]

po technické stránce mnoho zajímavých inovací jako velmi propracovanou funkci rozpoznávání a sledování 3D objektů a rozlišení jejich současného stavu nebo pokročilá práce i s více kamerami najednou. V podstatě se jedná o velmi dobrý příklad toho, čeho všeho je rozšířená realita schopná.

2.3.5 Wikitude

Wikitude je stejně jako Vuforia velmi profesionálně vytvořená knihovna, a i její použití je zpoplatněné, nicméně rovněž nabízí licenci zdarma na vyzkoušení, konkrétně na 45 dní s malým vodoznakem [28]. Důvod proč Wikitude vyniká je ten, že nabízí kompletní SDK řešení pro všechny podporované OS a pro více různých programovacích přístupů. Tuto již tak velkou nabídku navíc rozšiřují pluginy třetích stran [29]. Součástí Wikitude SDK je jak nativní, tak i JavaScriptové API pro Android a iOS, dále jdou k dispozici SDK pro Unity, Cordovu, Xamarin a Flutter. Pluginy pak rošiřují podporu o React Native a Ionic (framework pro tvorbu hybridních aplikací, podobně jako u Cordovy).

Navíc je k dispozici Wikitude Studio [30], což je vývojové prostředí, které umožňuje vytvářet jednoduché aplikace bez nutnosti psát jakýkoliv kód, pouze v prostředí 3D editoru. Zároveň pokud bychom chtěli jednoduše naši aplikaci publikovat, lze její obsah za měsíční poplatek zveřejnit v oficiální Wikitude aplikaci, kde stačí jen vyhledat danou ukázku a snímat k tomu určený tag. Protože tato knihovna pracuje jen s vlastními formáty pro obrázky a animace, Wikitude Studio na ně již vytvořený obsah umí převést.

2.3.6 ARToolKit (artoolkitX)

Je to jedna z nejstarších knihoven pro AR vůbec, vznikla již v roce 1999 a jako open source byla publikována o dva roky později [31]. Podporuje mnoho platforem, i těch desktopových, jmenovitě Windows, Linux, macOS a z mobilních Android, iOS a před pár lety i Symbian a Windows Phone. Knihovna je rovněž dostupná jako plugin do Unity.

Kvůli několikerým změnám v týmu vývojářů, a i vlastníků samotného projektu dnes probíhá primární vývoj pod názvem artoolkitX, nicméně existují i různé forky původního repozitáře ARToolKit, které prošly dalším nezávislým vývojem. Dnes je knihovna už upozaděna a existuje mnoho funkčnějších alternativ i přímo od tvůrců mobilních OS, ale díky své dlouhé historii a multiplatformnosti si místo v tomto textu zaslouží.

2.3.7 DeepAR

DeepAR je knihovna, s jejíž pomocí lze vytvářet filtry jako známe třeba z Instagramu [32]. Zaměřuje se tedy jen na augmentaci obličeje. Samotné SDK je uzpůsobené pro vývoj na Android i iOS zařízení přímo nebo za použití enginu Unity. S DeepAR lze i ve standardu HTML5 vytvořit plnohodnotnou webovou aplikaci a ukázky tohoto použijí ukazují vývojáři v dokumentaci. V SDK balíčku můžeme nalézt i již připravené masky a objekty použitelné jako filtr nebo jako výchozí bod pro další vývoj.

Nekomerční vyzkoušení knihovny je zdarma, ale v případě publikace je k dispozici ceník podle počtu aktivních uživatelů za měsíc. Do tisíce uživatelů je částka 250$ za rok, do pěti tisíc uživatelů to je 1000$ atd. až do individuálního cenění v případě nad sto tisíc aktivních uživatelů.

2.3.8 EasyAR

EasyAR je další knihovna pro AR, která primárně představuje placené řešení, tentokrát je ale možnost zkušební verzi nekomerčně publikovat s vodoznakem a bez většího omezení. Licence pro možnost i komerční publikace bez vodoznaku stojí 39$ na měsíc [33].

Knihovna umožňuje využívat pokročilé funkce jako sledování 3D objektu (nikoliv však jeho rozpoznání), hloubkové mapování prostředí, rozeznání a sledování obrázku nebo přehrávání videa s hardwarovou akcelerací. Vývoj je dostupný ve Swiftu nebo Objective-C na iOS nebo v Javě a Kotlinu na Android, případně lze knihovnu importovat do Unity.

2.4 ARCore

Od Androidu 7.0 a konkrétně od 1. března 2018 se Google stará o svou veřejně dostupnou knihovnu pro rozšířenou realitu, která není dostupná jen pro zařízení s OS Android, ale i jako webová aplikace [34]. Bez poplatků může kdokoliv vyvíjet a distribuovat aplikace, které za použití knihovny ARCore vytvořil. Problém nastává až u koncových zařízení, která musí tuto knihovnu od výroby podporovat a zároveň mít nainstalovanou nejnovější verzi aplikace Google Play Services for AR. Všechny

aktuálně podporované zařízení můžeme nalézt v oficiálním seznamu. Ze strany Googlu je to jakási snaha o zaručení funkčnosti a plynulosti přehrávání obsahu pomocí ARCore, certifikaci proto nemají starší a/nebo výkonnostně slabší zařízení, ale často i zařízení jen méně známých výrobců. U nově vyráběných telefonů je pouze menšina nepodporována, protože dnešní hardware je na to již dostačující i za nižší peníze a obecně převládá snaha tuto technologii podporovat, ale u dva a více let starých telefonů mají podporu většinou jen ty (ve své době) *high-endové* modely.

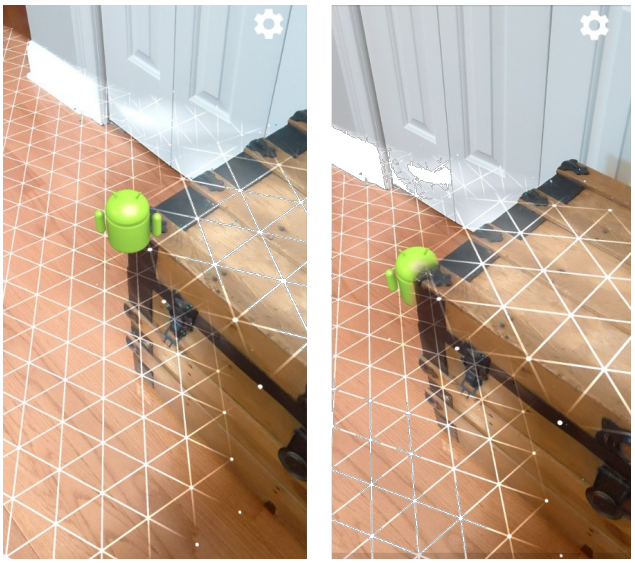
Sám jsem se přesvědčil o tom, že toto omezení někdy trochu pozbývá smysl. Samotný mechanizmus určování, zda-li je zařízení podporované nebo ne, je založený jednak na tom, že oficiálně v GP žádnou aplikaci označenou jako ***AR Required*** *nenalezneme, a jednak že Google Play Services for AR obsahují seznam podporovaných zařízení a systémový otisk vašeho zařízení vás usvědčí, což způsobí pád i aplikací získaných stažením externího .apk balíčku. Je přesto možné toto omezení v některých případech obejít, ale není to postup vhodný pro běžného uživatele, vyžaduje totiž root telefonu a aplikaci Magisk. Práce samotná jednak vyžaduje jisté zkušenosti a rovněž hrozí riziko nevratného poškození hardwaru i softwaru telefonu. Magisk obsahuje modul jménem ARCore/Playground Patcher. Tento modul funguje tak, že přepíše systémový otisk vašeho telefonu na telefon Google Pixel 2, což je oficiálně podporované zařízení knihovnou ARCore. Poté máte možnost si Google Play Services for AR oficiálně z GP stáhnout, a i aplikace vyžadující ARCore začnou být v GP k dispozici. Tento postup mám odzkoušený na zařízení Asus ZenFone 5 (které podporováno není), ale nedosáhl jsem úspěchu vždy. Většina aplikací vyžadující ARCore fungovaly bez problémů, ale pár jich hned po zapnutí přestalo fungovat. Nicméně pokud se aplikace úspěšně spustila a nespadla hned ze začátku, i na zmíněném zařízení lze používat rozšířenou realitu s ARCore naprosto plynule. Vinu v tomto je třeba přičítat spíše samotným výrobcům telefonů, že nemají zájem certifikaci pro použití ARCore vůbec na své produkty získat, i když by na to hardwarově stačily. Jak jsem ale uvedl výše, dnes je úroveň podpory na mnohonásobně lepší úrovni.*

Obecně Google rozděluje základní principy funkčnosti do několika skupin, které dohromady utváří prostředí rozšířené reality:

Motion tracking (Sledování pohybu) Sledují se body (tzv. *feature points*), jak lze vidět na ilustračním obrázku **2.6**, dobře rozpoznatelné ve svém okolní vlivem své odlišné barvy, kontrastu, nebo hrany představující přechod mezi plochami stejného objektu, či více různých objektů. Při sledování více různých bodů zároveň lze poznat vzdálenost mezi nimi a vlivem překrývání i možné překážky.

Environmental understanding (Porozumění prostředí) U prostředí je důležité určit přibližný tvar a detekovat roviny (tzv. *planes*) spolu s jejich hranicemi. Detekované roviny lze použít k ukotvení virtuálního objektu.

Depth understanding (Porozumění hloubky) Tato funkce není klíčová, ale utváří ještě lepší dojem z rozšířené reality. Umožňuje aplikaci poznat, že při posunu v prostoru je část detekované roviny schována za jiným objektem, který tam z jiného úhlu pohledu nevadil. Tímto se i



Obrázek 2.5: Ukázka funkčnosti na zařízení bez (vlevo) a s Depth API (vpravo), převzato z [35]

část virtuálního objektu přestane zobrazovat, protože by správně neměla jít vidět (pro představu viz obrázek **2.5**). Práci s hloubkou umožňuje Depth API, což je ale záležitost konkrétního telefonu a ne všechny ho podporují, nicméně pro práci ARCore to není nutná funkcionalita.

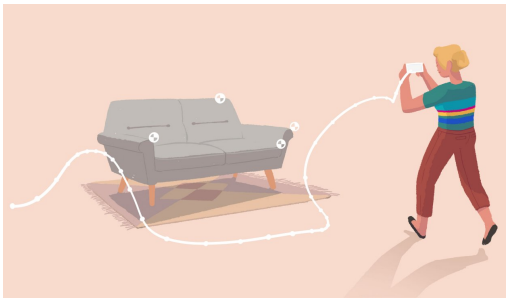
Light estimation (Odhad světla) Pohybem při mapování okolí si ARCore všímá i měnících se odrazů a odlesků. To umožňuje přibližně určit polohu zdroje světla a tím vliv světla a stínů přidat i do virtuální scény, jak můžeme vidět v ukázce **2.7**. Je to ale poměrně náročné na výkon zařízení u složitějších modelů.

User interaction (Interakce uživatele) Pro věrnější zážitky je cílem AR i nějak prostředkovat interakci uživatele s prostředím. Díky všem předchozím funkcím je tak možné z dotyku 2D obrazovky vést přímku pod úhlem telefonu v prostoru a najít příslušný průsečík s detekovanou rovinou. Poté záleží na vývojáři, jak se tato interakce projeví.

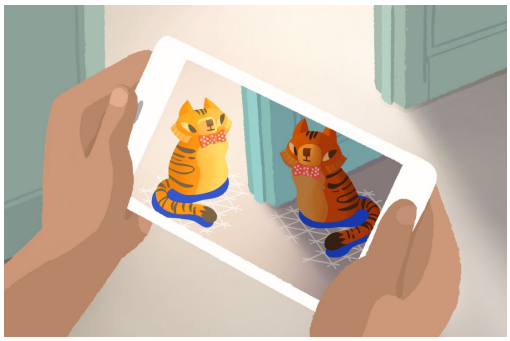
Oriented points (Orientované body) Tato funkce umožňuje mapovat i nakloněné roviny a nerovné povrchy a pracovat s nimi.

Anchors and trackables (Kotvy a sledovatelné objekty) Pro umístění virtuálního objektu do scény je třeba pro něj vytvořit kotevní bod, jehož pohyb a vzdálenost od zařízení bude určovat, v jakém natočení a v jaké vzdálenosti od zařízení se má objekt renderovat. Chceme nejčastěji aby byl položený na zemi.

Augmented Images (Rozšířené obrázky) Tato funkce umožňuje rozpoznávat předem definované obrázky (tagy) a na jejich plochu promítat virtuální scénu. Pro tuto funkci není třeba zkoumat okolní prostředí tagu, proto je velmi rychlá.



Obrázek 2.6: Motion Tracking s feature pointy, převzato z [36]



Obrázek 2.7: Light Estimation, převzato z [36]

Sharing (Sdílení) Cloud Anchor API umožňuje sdílet jedním zařízením vytvořené kotevní body mezi ostatními v téže lokaci. Uživatelé tak mohou sdílet stejnou scénu, pokud se dívají na stejné místo a stačí, že pouze jeden z nich provedl důkladnější mapování.

2.4.1 Mapování okolního prostředí

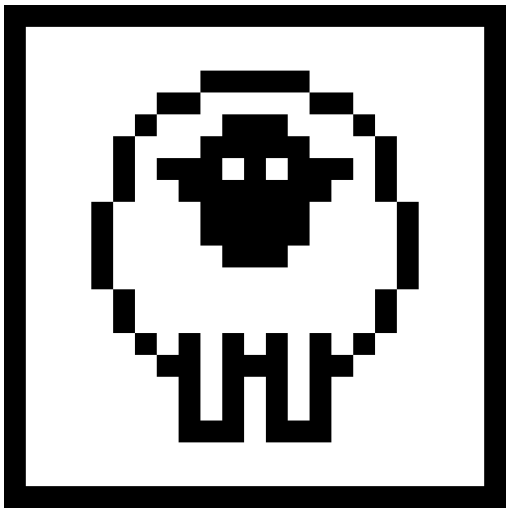
Pokud chceme umístit předdefinovaný model do námi určeného prostoru je třeba nejprve zmapovat nejbližší okolí. Takto zprostředkujeme informace o tom, kde se nachází objekty (podlaha, stěny, nábytek atd.) v okolí, ty budou sloužit jako záchytné body pro snazší orientaci v prostoru i při pozdějším pohybu. Knihovny pro rozšířenou realitu v reálném čase zpracovávají data o pohybu zařízení z akcelerometru a fotoaparátu. [37, 38]

Akcelerometr měří zrychlení pohybu v jednotlivých osách kartézské soustavě, tím určuje nejen směr pohybu, ale i uraženou vzdálenost v prostoru. Dále jsou získávána obrazová data ze zařízení pomocí fotoaparátu a ta jsou analyzována. Díky drobným rozdílům v těchto snímcích vlivem pohybu zařízení lze detekovat míru posunu objektů a deformace okolí. Takto se s přibývajícími daty čím dál přesněji odhadují tvary jednotlivých objektů, jejich vzdálenosti od sebe a rovněž jejich vzdálenost od zařízení. ARCore umí využít Depth API pokud je zařízením podporováno a tím zpřesnit a zrychlit určování vzdáleností. Zároveň se lépe tvoří hloubkové mapy okolí. [39]

Cílem je především detekovat roviny a větší předměty, které jsou vhodné k vytvoření kotev. Ideální k tomu jsou podlahy s decentním vzorem nebo kobercem, větší předměty barevně odlišitelné od okolí nebo stěny třeba s obrazy. Obecně lze říct, že vysoké kontrasty usnadňují mapování, a naopak velké jednobarevné plochy bez vzorů jsou pro funkčnost problematické.

2.4.2 Ukotvení modelu do reálného prostředí

Ukotvení je proces zvolení bodu detekované roviny, na který chceme umístit virtuální objekt. Jak se potom bude naše zařízení pohybovat, virtuální objekt bude následovat pohyb svého kotevního bodu. Ve výsledku máme téměř dokonalou iluzi toho, že virtuální předmět jakoby do reálného prostředí patřil.



Obrázek 2.8: Ukázka vhodného vzhledu AR tagu

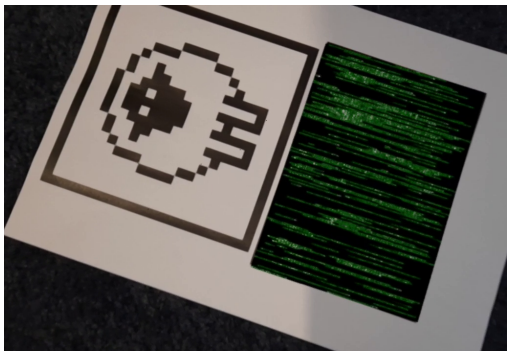
Může se stát, že kotevní bod bude vlivem našeho pohybu nějaký čas mimo zorné pole zařízení, to nutně nemusí vadit, protože plocha již byla detekována a globální souřadnice kotevního bodu dočasně uloženy. Díky datům o poloze a rotaci zařízení se ví, kde by se měla kotva nacházet vzhledem k zařízení, a když se znovu objeví v záběru, může se virtuální objekt znovu začít promítat. Problém nastává většinou až ve chvíli, kdy se například rychlejším pohybem dostaneme do částečně nebo zcela nezmapovaného území, tím je třeba proces mapování provést znova, a buďto v rámci úspory paměti se smažou data o předchozích oblastech, nebo se knihovně nepodaří obě oblasti propojit kvůli odlišně odhadnutým vzdálenostem.

2.4.3 Detekce AR tagů

AR tagy jsou jednoduché značky, které mají funkci pomocníka při utváření rozšířené reality. V ideálním případě jsou tvořeny jednoduchým, ale unikátním a špatně zaměnitelným vzorem s vysokým kontrastem, viz **2.8**. Může se jednat o např. QR kód nebo čárový kód, ale přijatelný je i velký text. Jako AR tag lze pochopitelně nastavit úplně jakýkoliv obrázek, ale je třeba si uvědomit, že vnímání barev se velice mění vlivem intenzity a barvy dopadajícího světla, tím pádem nalezení barevného tagu může být výrazně zdlouhavější. Stejné problémy může způsobovat i různé podání barev a kvalita snímků napříč telefony od různých výrobců [40]. Vhodnost snímku si lze i ohodnotit podpůrným nástrojem [41].

Důvod používání AR tagů je ten, že na vytvoření kotevního bodu pro položení modelu není třeba mapovat okolní prostředí. Jako kotva se použije samotný tag, což velice urychluje celý proces, zároveň se při tvorbě obsahu spojeném s daným tagem může počítat i s okolním prostředím. Jelikož aplikace má své spouštěcí tagy uložené, stačí je jen analýzou obrazu vyhledávat. Vnější obvod tagu bude mít nejčastěji tvar obdélníka, ale vlivem perspektivní deformace se bude z pohledu zařízení

Obrázek 2.9: Detekování tagu ve formě textu



Obrázek 2.10: Při otočení tagu se video přizpůsobí

jevit jako lichoběžník nebo jen obecný čtyřúhelník. Jelikož zařízení o své poloze a natočení v prostoru ze senzorů představu má, může tak z deformace tagu určit natočení roviny tagu a tomu přizpůsobit promítání virtuálního objektu, to je ukázáno na obrázcích (**2.9** a **2.10**). Na rozdíl od mapování okolí **2.4.1 (Mapování okolního prostředí)** zde teoreticky stačí analýza pouze jednoho snímku, ale poté se může umístění objektu jevit mírně nestabilní a uskákané.

2.5 OpenGL

OpenGL je velmi populární multiplatformní a multijazykové vysokoúrovňové API zprostředkovávající 2D a 3D rendering [42]. První verze vznikla už v roce 1992 a od té doby postupně přibývaly i mírně odlišné verze pro jiné využití. Nejpoužívanější z nich je OpenGL ES, což je verze pro *enbedded zařízení*, ale hlavně pro Android zařízení a to již léta ve verzi 3.1 [43]. V současnosti se o vývoj s veřejným zdrojovým kódem stará nezávislá skupina Khronos, ale poslední verze je z roku 2017. Popularita OpenGL postupně padá a hledají se alternativy, mnoho nových API ale z něj stále vychází. Apple již mnoho let využívá své vlastní nízkoúrovňové API Metal a staví na něm svou platformu. Obecná náhrada by mělo být nízkoúrovňové API Vulkan [44], které již má i podporu na Androidu, ale zatím tak populární není. Mimo Apple v současnosti nejsou plány na kompletní náhradu OpenGL jiným API.

2.6 Filament

Toto renderovací API pracuje na principu fyzikálního renderování, což je přístup zaměřující se na tok světla a jeho chování při kontaktu s materiály. Filament je vytvořen pro velmi široké spektrum použití, protože vychází z velkého množství backendových API pro různé platformy a využití [45]:

- OpenGL 4.1+ pro Linux, macOS a Windows
- OpenGL ES 3.0+ pro Android a iOS

- Metal pro macOS a iOS
- Vulkan 1.0 pro Android, Linux, macOS, a Windows
- WebGL 2.0 pro všechny platformy

2.7 Sceneform

Sceneform je vysokoúrovňové renderovací API založené na Filamentu. Jeho rozsah využití je velmi úzký a to konkrétně pro zjednodušení práce s rozšířenou realitou a konkrétně knihovnou ARCore. Celkově je práce s AR o mnoho jednodušší a potřebný kód mnohonásobně kratší než za použití klasického OpenGL [46].

Dlouhou dobu vycházely nové verze ARCore a Sceneformu spolu. Ve verzi 1.16 byl vydán Sceneform jako open source a přinesl soustu nových funkcí, jako například podpora formátů .gltf a .glb ze strany Filamentu. Problém nastal s verzí 1.17 kde vývojáři ohlašovali zásadní problémy i se základní funkčností. Ještě větší problém nastal, když Google oznámil, že už dále ve vývoji nehodlá pokračovat a jako hot fix vydal verzi 1.17.1, která je ale totožná s verzí 1.15. Tímto byl zároveň repozitář na GitHubu označen jako *archived* a práce na něm skončily [47].

Naštěstí zafungovala skvělá Android komunita a objevil se schopný vývojář jménem Thomas Gorisse, který se o svůj fork originálního repozitáře neustále stará a vyvíjí ho dál na svém GitHubu [48]. Velmi pružně reaguje na veškeré dotazy a problémy vývojářů a do Sceneformu vždy přinese podporu nejnovější vydané verze Filamentového backendu nebo zkontroluje plnohodnotnou funkčnost s nejnovější verzí ARCore.

Musím zdůraznit, že právě tento repozitář, a hlavně kvalita podpory ze strany Thomase Gorisse byl zásadní důvod, proč jsem jako renderovací API zvolil Sceneform. Nicméně i původní Sceneform ve verzi 1.15 i 1.16 je stále kompletně funkční řešení a ani u nejnovějších verzí ARCore jsem s nimi nenarazil na žádný problém. Nicméně oficiální vývoj již neprobíhá a žádná náhrada nebyla dosud oznámena.

2.8 Licence

Vývoj aplikací na Android není zpoplatněn, každý si může zdarma stáhnout třeba Android Studio pro svou distribuci OS a začít s tvorbou. Velmi podobně to platí i pro ostatní vývojové nástroje. Díky tomu vzniká nepřeberné množství návodů, diskuzních fór a projektů, což velmi zjednodušuje dostupnost znalostí pro vývoj na Android začínajícím i pokročilým vývojářům.

Vývoj na iOS je pro soukromé účely možný zdarma a všechny prostředky jsou na to dostupné rovněž zdarma. Zdarma ale není možnost aplikace publikovat oficiální cestou přes obchod App Store. Je třeba mít aktivní vývojářský program Applu [49], což je zpoplatněná služba za 99$ na osobu za rok [50].

Kapitola 3

Tvorba animací

Tato část se zabývá tvorbou 3D modelů s animacemi, které budou vhodné pro použití v AR. Důraz není tolik kladen na detailnost zpracování, protože to s sebou přináší i potřebu vyššího výkonu zařízení, na kterém se animace spouští. Snaha je tedy modely a jejich části vytvářet jako low-poly modely (tvořené nízkým počtem polygonů), což velice zjednodušuje práci s nimi a zároveň jsou kladeny menší nároky na úložný prostor.

Další podkapitoly se zabývají konkrétními prostředky a praktikami, jak takové modely co nejefektivněji vytvořit a přidat k nim nejrůznější animace. Rovněž lze vycházet z již hotových a animovaných scén, které bychom chtěli vhodně upravit do jiné podoby, čehož bylo často využíváno v části **4 (Praktická část)** této práce.

3.1 Cinema4D

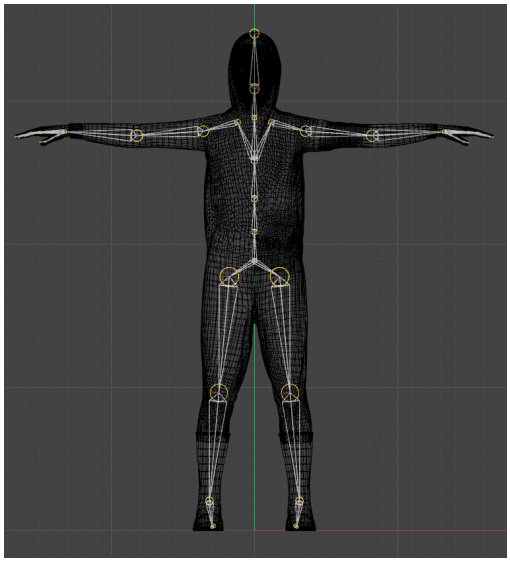
Jedná se o velmi mocný nástroj používaný k tvorbě velmi propracovaných modelů i animací [51]. Je využíván profesionálními filmovými studii na tvorbu speciálních efektů mnoha známých snímků jako třeba Počátek nebo Tron: Legacy. Rovněž byl použit pro tvorbu mnoha kompletně animovaných filmů jako např. Polární expres nebo Beowulf.

Tento program je vyvíjen firmou Maxon již od roku 1990. V současné době je dostupný na platformách Windows, macOS a ve formě pouze renderovacího klienta přes terminal i na Linux. Jedná se o placený software, který ovšem nabízí za symbolický poplatek možnost studentské licence (více informací v podkapitole **3.7**).

Od roku 2010 podporuje Cinema4D psaní vlastních funkcí pomocí skriptovacího jazyka Python, tímto je možno relativně snadno modifikovat modely způsobem, který by pomocí čistě jen standardních funkcí nebo pluginů mohl zabrat mnoho hodin. Během praktické části této práce ale skriptování nebylo zapotřebí.

Pro tvorbu statických scén program obsahuje veliké množství předdefinovaných objektů, které lze tvarově upravovat základním ovládáním (změna velikost, rotace a pozicování) nebo aplikováním

Obrázek 3.1: Ukázka T-pózy, převzato z [52]



Obrázek 3.2: Rig postavy

deformačních funkcí. Další možnosti úprav se zpřístupňují převedením objekt na polygony, díky tomu lze provádět deformace na úrovni bodů, hran nebo ploch. Při práci s polygonovými objekty namísto předdefinovanými je výsledný soubor projektu zpravidla větší, práce s ním je výpočetně náročnější a aplikování některých deformací může u objektů s více polygony trvat značně déle. Některé funkce ale s jinými, než polygonovými objekty pracovat nedokáží. Např. při *rigování postav* (kterému se věnuje podkapitola **3.2**) a jejich animování musí být kůže tvořena polygonovou sítí.

Pro dobrou orientaci ve scéně nám může pomoct změna pohledu, na výběr je klasické 3D zobrazení a další tři okna nám umožňují dívat se na scénu skrze rovinné řezy (rovinami XY, YZ a XZ), které nemají deformaci způsobenou hloubkou a vzdáleností od pomyslné kamery. Ve výchozím nastavení se pro 2D zobrazení používá drátěný model, kde jsou viditelné pouze hrany.

3.2 Rigování postav

Rigování je velice důležitý proces během tvorby animace pohybu postavy. Při něm se postavě, která je nejčastěji vytvořena v T-póze (ukázka na obrázku **3.1**), přidávají vlastnosti kostí a kloubů. Pokud chceme, aby animace člověka při chůzi vypadala realisticky, je třeba určit kde v těle se mají nacházet místa ohybu (klouby) a kde naopak by ohyb neměl být možný (kosti). K tomuto procesu je vhodné mít alespoň základní představu o anatomii člověka. Hotový rig je pro ilustraci ukázán na obrázku **3.2**.

Cinema 4D nabízí pomocníka při tvorbě rigu postavy (objekt Postava), který vytvoří základní kostru s průměrnou tělesnou konstitucí, tu je ale třeba většinou upravit pro potřeby konkrétní postavy. U takto vytvořené kostry je poté přidána závislost na polygonové síti postavy, a pak můžeme spustit předdefinovanou animaci chůze. Uvidíme tak práci všech kloubů a kostí a můžeme

případně jejich umístění upravit, aby animace vypadala správně. Výhoda takto vytvořeného rigu je, že s ním lze celistvěji manipulovat, pokud bychom chtěli postavu dostat do kleku, stačí jen posunout pánevní část dolů, v případě kloubů rigu bychom museli všechny správně posunout a ohnout. Takovéto úpravy lze kombinovat i s již existující animací.

Rigování lze provést i postupným přidáváním kloubů a kostí a napojováním je na sebe, takový postup je obecný a je vhodný, pokud chceme soubor s animací exportovat mimo Cinemu4D a záleží nám na zachování všech animací. Úplně stejný způsob lze použít i u animací pohybu nebo ohýbání jakýchkoliv objektů tvořených polygony. Propojení objektu (třeba postavy) s kostrou poté probíhá tak, že je třeba k polygonovému objektu vytvořit objekt ***Kůže***, který bude mít závislost nastavenou na všechny potřebné klouby a kosti, poté přidáme na polygonový objekt tag ***Vliv***, který nastavíme na předtím vytvořenou Kůži. Poté si celý proces můžeme ověřit nastavením libovolných klíčových snímků (více informací v podkapitole **3.5**).

3.3 Základy tvorby animací

V grafice se pohyb tvoří rychlým střídáním statických obrázků a díky nedokonalosti lidského oka výsledný vjem vypadá jako plynulý pohyb, čehož se využívá již od počátků kinematografie. Tvorba animací je založen na úplně stejném principu. V jeden konkrétní okamžik je objekt na určitém místě vyjádřeném souřadnicemi a v každém pozdějším okamžiku se o kousek posune (případně ohne, zdeformuje, zmenší apod.). Výsledkem spojení snímků, pořízených v jednotlivých okamžicích je pak videosekvence. Frekvence pořizování snímků může být různá, obecně se 24 snímků za sekundu považuje za plynulý záznam (TV vysílání nebo kino) [53], hráči videoher ale obvykle považují jakoukoliv hodnotu pod 60FPS za nedostačující.

Při tvorbě animace je důležité definovat pozici všech bodů ve scéně pro každý snímek na časové ose. Mnohdy nám pomůže, že ne všem bodům se pozice tak často mění, přesto ale tento proces dokáže být velmi zdlouhavý. Proto existují různé metody pro zjednodušení animování. Mezi nejznámější patří ***Keyframing* 3.5** díky své celkové jednoduchosti, pro velmi přesné a realistické pohyby nejčastěji zvířat a lidí se používá ***Motion capture* 3.4**. Cinema4D nabízí rovněž usnadnění animace chování předmětů, na něž působí přírodní síly reprezentované funkcemi proměnných parametrů. Nejčastěji se může jednat o gravitaci, vítr nebo působení jiných předmětů při kolizi, a výsledné chování se pak odvíjí podle nastavení materiálů.

3.4 Motion capture

MoCap technologie slouží k detailnímu zachycení pohybu osob a zvířat. Sledovat lze buďto tělo jako celek zachytáváním pohybu určitých bodů, nebo se zaměřit na sledování jen určitých částí, například obličejové mimiky.

Obrázek 3.3: Vzor teček technologie FaceID zachycený IR kamerou, převzato z [56]

Nejjednodušší způsob sledování pohybu je použitím hloubkového senzoru, ten je schopen rozeznat hlouku i kontury objektů. K tomu se používá infračervené záření: do pozadí se vyšle vzor teček a infračervená kamera snímá jejich prostorem zdeformovaný zpětný odraz [54] na jehož základě se určí tvar a vzdálenost věcí v okolní [55]. V praxi tento přístup můžeme najít v zařízeních Kinect od společnosti Microsoft nebo jako součást technologie FaceID společnosti Apple na rozpoznávání obličeje, viz **3.3**. Čím blíže se objekt nachází k hloubkovému senzoru, tím je detekce tvaru přesnější.

Pokročilejší přístup spočívá v nasazení mnoha senzorů pohybu na tělo subjektu. Tyto senzory jsou schopny určit směr a rychlost pohybu a zároveň rotaci v prostoru. Tělo člověka osadíme těmito senzory na místech, odpovídající rigu virtuální postavy, kterou chceme animovat, tím je možné jeho pohyb nahrát pro potřeby animace. Na rozdíl od předchozí metody zde není žádná závislost přesnosti měření na vzdálenosti senzorů od sebe, je ale třeba jejich vzájemnou polohu znát.

Nejpřesnější metoda vyžaduje množství konkrétně napozicovaných kamer v prostoru, které nahrávají vymezený prostor každá z jiného místa. Sleduje se pohyb bodů nebo vzorů umístěných na těle subjektů společnou analýzou záběrů. Tato metoda je využívaná filmovými (viz **3.4**) i herními studii pro detailní záznam pohybu a následnou tvorbu animací lidí nebo zvířat [55].

3.5 Keyframing

Keyframing je technika tvorby animací pomocí definování klíčových snímků, ty ukládají data buďto o pozici bodů objektu, které se v čase mění nebo míru aplikované deformace. Deformace může být určena definovanou funkcí, posunem objektu, ohybem přidruženého kloubu, či rotací. Celé zjednodušení tvorby animací tímto způsobem spočívá v tom, že není nutné, aby každý snímek animace byl

Obrázek 3.4: Herec Andy Serkis při natáčení filmu Star Wars: Síla se probouzí převzato z [57]

takto nastaven, ale prostředí samo utvoří plynulý přechod z jednoho stavu do druhého. Aby byla animace plynulá, pohyb začíná postupně zrychlovat a od doby přesně mezi dvěma klíčovými snímky naopak zpomaluje. Tím se animační prostředí snaží více přiblížit reálně vypadajícímu pohybu.

Vzhled animace je plně v rukou animátora, ale dosáhnout tímto způsobem reálně vypadajících animací pohybu postav je složitý a hlavně zdlouhavý proces. Pro úpravy vlivu funkcí, posun předmětů nebo animování jednoduchých struktur je metoda zcela dostačující.

3.6 Další dostupný software a služby

Pochopitelně existuje nesčetné množství alternativ pro tvorbu prvotřídních 3D modelů a animací pro různé platformy. Každá má mírně jiný okruh specializace, a proto vzájemné srovnání není tak důležité, tato sekce se proto spíše zabývá jejich využitím. Nejčastějším společným limitem bývá dostupnost licence a, jak bývá zvykem u profesionálního softwaru, velmi vysokou částkou. Pro představu zde uvádím tabulku cen za roční předplatné **3.1**.

3.6.1 Blender

Díky tomu, že Blender je jako jeden z mála programů svého typu dostupný zdarma, je i velmi populární [67]. Je vyvíjen s veřejným zdrojovým kódem dostupným v repozitáři na GitHubu [68], což pomáhá tvorbě nejrůznějších pluginů, které mohou velmi zjednodušit práci. Komunita designérů se rovněž podílí na vzniku velkého množství naučných a ukázkových videí své tvorby nebo diskuzních

Tabulka 3.1: Ceník animačních softwarů

Software	**Cena za rok v USD**	**Studentská licence**	**Podmínky požití volné/studentské licence**
Cinema4D [58, 59]	719	Ano	6 měsíců Nekomerční použití
Blender [60]	0	Ne	-
Autodesk Maya [61, 62]	1700	Ano	1 rok Nekomerční použití
Autodesk 3ds Max [61, 63]	1700	Ano	
Unity [64, 65]	0/399/1800	Ano	Za uplynulý rok výdělky nižší než 100 tisíc $
Unreal [66]	5% při výdělcích nad 1 milion	Ne	-

fór. I přes jeho popularitu ale obecně převládá názor, že použití Blenderu je velmi složité a úplní začátečníci jsou do značné míry závislí na vytvořených návodech, a ne na své intuici.

3.6.2 Autodesk Maya

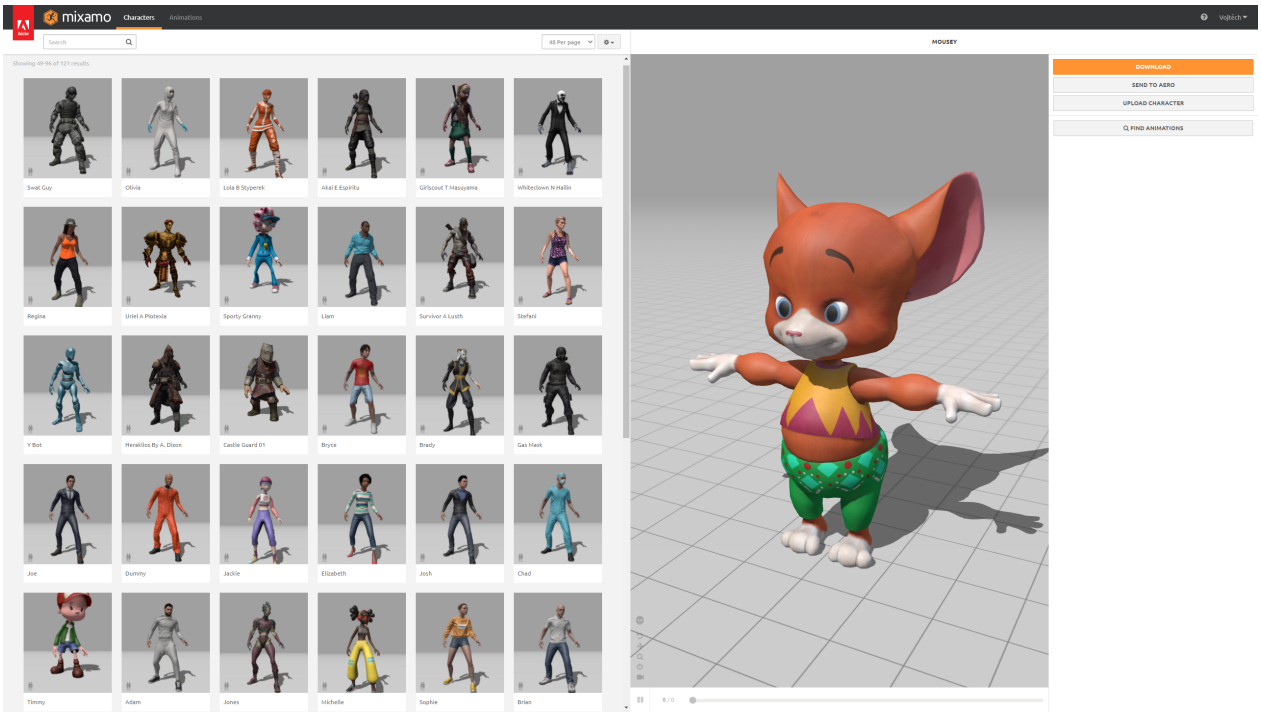
Maya je software dostupný na všech hlavních platformách jako zkušební verze zdarma na 30 dní nebo placená plná verze. Možnosti má velice podobné jako Cinema4D a stejně tak se používá na profesionální tvorbu animací i ve filmovém průmyslu [69]. Jako hlavní formát je používán .fbx, která se díky dostupnému SDK Autodesku stal obecně používaným standardem při exportu mezi různými softwary.

3.6.3 Autodesk 3ds Max

Jedná se o software primárně určený pro tvorbu vizualizací vnějších i vnitřních prostor budov, domů nebo bytů [70]. To jeho použití pro tvorbu animací ale nijak neomezuje jen na tuto oblast. Nabízí výbornou podporu exportu mezi produkty firmy Autodesk, což kvůli převážně architektonickému využití 3ds Maxu je důležité hlavně u softwaru AutoCAD. Vyvíjen je ale pouze pro OS Windows.

3.6.4 Mixamo

Tato služba od společnosti Adobe je dostupná zdarma [71] na webové stránce `http://mixamo.com/` po přihlášení. Nabízí možnost si vybrat z široké nabídky profesionálně vytvořených postav na které lze aplikovat nejrůznější animace pohybu **3.5**. Tyto animace jsou rovněž součástí Mixama a lze je pomocí přednastavených parametrů upravovat. Například když postava nese tašku, lze nastavit její domnělá váha, při animaci skoku můžeme změnit jeho délku, u tance pohybový rozsah paží a tak dále. Takto vytvořené animace se výborně hodí jako základ pro naučení se práce s rigem, pro začínající grafiky nebo tvůrce počítačových a mobilních her. Cinema4D dokáže Mixamo modely i



Obrázek 3.5: Nabídka Mixamo modelů a animací, převzato z [72]

s rigem a animacemi kompletně importovat díky přímé podpory ze strany Adobe a tím umožnit a usnadnit další úpravy.

3.6.5 Unity a Unreal

K tvorbě animací lze využít i volně dostupné grafické enginy jako právě Unreal a Unity, pravděpodobně by ale nikdo nepoužíval tak mocné nástroje jen k tomuto účelu. Pokud bychom brali v potaz využití enginů jen na tvorbu her, tak zde můžeme animace najít v tzv. *cutscénách*, tedy scénách, kde se předem určeným způsobem postavy pohybují a většinou je vyprávěn příběh. Často je ale ve hrách možná úprava vzhledu postavy (pohlaví, oblečení, výbava atd.), a proto musí být animace připravena obecně na rigu nebo základní postavě s možností přidání oblečení až během přehrávání. Vytvářet takto variabilní animace jiné softwary neumožňují, ale ani to není jejich primárním cílem.

3.6.6 Houdini

Jedná se o velmi profesionální software, který je hojně využíván filmovými studii na vytváření speciálních efektů nebo kompletně animovaných celovečerních filmů [73]. Paralelně je k dispozici i zdarma dostupná verze [74], která lze využít k nekomerčním účelům, ale neposkytuje všechny možnosti plné verze a rendering je možný jen v nízkém rozlišení s vodoznakem. K dispozici je na OS Linux, macOS a Windows.

3.7 Licence a publikace

Kvůli tomu, že Cinema4D je velmi profesionální animační nástroj, lze pochopit, že jeho pořizovací náklady nejsou zanedbatelné, viz **3.1**. Pro potenciální zájemce je k dispozici třicet dní zkušební doby, kdy funkčnost programu není nijak omezena a je tedy možnost si vyzkoušet, jestli se investice vyplatí.

Pro účely této práce je ale důležité, že Maxon poskytuje licence pro učitele a studenty [58] a rovněž umožňuje nákup multilicence pro vzdělávací instituce. Takovéto licence nemají žádné omezení, co se týká dostupných funkcí, jsou omezeny pouze možnostmi publikace výtvorů. Není možné takovouto licenci využívat ke komerčním účelům, ale pouze k osobním a vzdělávacím. Studentská licence využitá v této práci byla získána na základě předložení potvrzení o studiu a uhrazení administrativního poplatku 3,33€. Licence je platná 6 měsíců ode dne ověření.

Kapitola 4

Praktická část

Tato kapitola má za cíl přiblížit a vysvětlit zvolené postupy, které byly během tvorby aplikace pro rozšířenou realitu použity. Jak již je zřejmé z obsahu předchozích kapitol, v aplikaci se vyskytují 3D modely a animace. Některé z nich jsou veřejně dostupné, jiné byly pro účely aplikace zakoupeny a nebo jsem použil své již dříve vytvořené modely.

Během příprav a vývoje jsem narazil na mnoho problémů, které bylo třeba vyřešit a překonat. Těm bude rovněž věnována pozornost a tento text poté může sloužit jako návod nebo opora pro další vývojáře a pomoct jim tak se stejným problémům vyhnout nebo s nimi alespoň počítat.

Aplikace, která vznikla pro účely této bakalářské práce bude sloužit jako výchozí bod pro následné rozšíření o obsah bližší použití na hradě Sovinec. Tam se nepředpokládá využití všech funkcí této aplikace, ale hlavně zobrazování animací na detekovaném tagu.

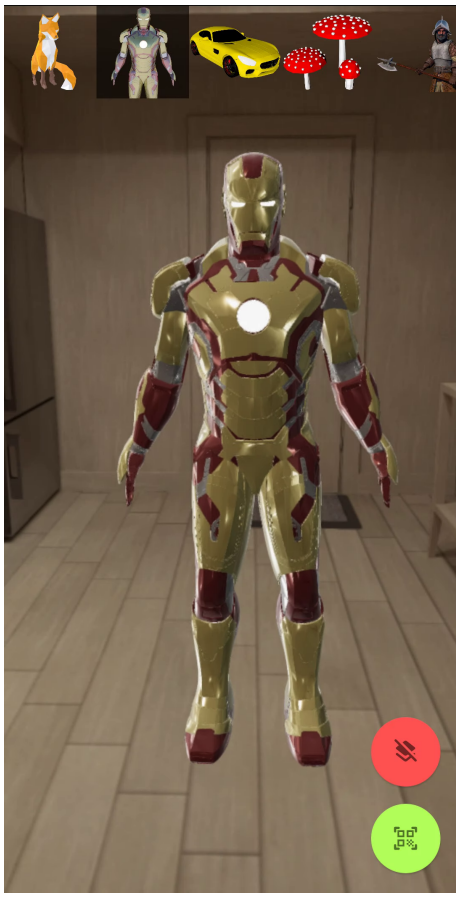
4.1 Uživatelské rozhraní a ovládání aplikace

Obsah aplikace by měl co nejvíce zaujmout uživatele a svým způsobem ho vtáhnout do děje. Toho lze dosáhnout především kvalitně vytvořeným a příjemně vypadajícím prostředím. Jelikož je cílem nabudit dojem, že rozšířená realita je skutečná, vhodné je zprostředkovat vstup do ní nerušeně. To znamená že veškeré nastavení a prvky GUI by měly být co možná nejvíce skryty abychom efektivně využili potenciál velikých obrazovek dnešních telefonů, případně tabletů.

Při úplně prvním spuštění aplikace je uživatel systémem vyzván, aby povolil využívání fotoaparátu, což je pochopitelně nezbytné pro používání aplikace. Povolení lze udělit buď jednorázově, tím pádem při dalším spuštění bude třeba jej udělit znovu, nebo trvale, což bude platit do doby, než bychom smazali data a mezipaměť aplikace.

Pokud na telefonu není nainstalovaná nejnovější verze podpůrné služby Google Play Services for AR[1], je uživatel přesměrován na GP aby ji aktualizoval nebo nainstaloval. I když by vyšla nová verze této služby, má aplikace by ji nevyžadovala. Musel bych vydat novou verzi, kde bych buďto já

[1] V době psaní tohoto textu je nejaktuálnější verze ta s číselným označením 1.23



Obrázek 4.1: Ukázka módu aplikace pro výběr a umístění modelu do prostoru

Obrázek 4.2: Ukázka módu aplikace pro detekci AR tagu

změnil nastavení v *build.gradle* souboru na nejnovější verzi ARCore, nebo bych musel aktualizovat závislost aplikace na novou verzi Sceneformu, která by udělala to samé, ale interně v rámci svého SDK. Dokud je zajištěná zpětná kompatibilita mezi verzemi ARCore, tak to většina vývojářů AR aplikací řeší tak, že nevydávají zbytečné aktualizace, které by stejně nepřinesly nic nového jejich aplikaci, ale pouze vyžadovaly novější verzi ARCore.

Po úspěšném spuštění lze vidět, že aplikace zobrazuje to, co vidí hlavní kamera telefonu. Obraz částečně překryje piktogram ruky držící telefon, který se krouživě pohybuje po obrazovce. Přesně tento pohyb se očekává od uživatele, aby bylo možné zmapovat okolí. Když už byla detekována alespoň jedna plocha, obrázek ruky zmizí a na dané ploše se začnou objevovat pravidelné tečky. Ty označují místa, kam je možné umístit nějaký model.

Kromě obrazu z kamery si lze všimnout dvou tlačítek v pravém dolním rohu obrazovky (*FloatingActionButton*). Jedno odstraňuje ze scény všechny doposud položené modely a druhé přepíná mód aplikace z prostého umisťování modelů (viz obrázek **4.1**) do módu skenování AR tagů (viz obrázek **4.2**). Zároveň v prvním zmíněném módu je nahoře k dispozici horizontálně scrollovací na-

bídka modelů (*HorizontalScrollView*), které lze vybrat a umístit. Ve druhém módu je zorné pole zmenšeno a očekává se, že se uživatel přiblíží k tagu tak blízko, aby jím zaplnil co největší část vymezeného prostoru[2]. V případě, že je tag skutečně nalezen a rozpoznán, uživateli je zobrazena hláška o probíhajícím načítání a je přenesen zpět do prvního módu. Načítací hlášku je podle mne vhodné zobrazovat, protože než se model skutečně objeví zpravidla chvíli trvá a během této doby mohou být odezvy telefonu zpomalené, uživatel tak snáze pochopí, že se na pozadí něco děje a že je toto chování správné. Doba mezi detekcí tagu a zobrazení tomu odpovídajícího obsahu se pochopitelně odvíjí od velikosti načítaného souboru.

Aby zobrazovací prostor byl co největší tak se během startu aplikace skryje *Action bar* aplikace. Zároveň se aplikace spouští v režimu *fullscreen*, tím pádem se Android sám postará o skrytí horní notifikační lišty a spodní navigační lišty, dokud je uživatel tahem prstu od okrajů obrazovky nevyvolá. I v takovém případě se ale samy zase po chvíli schovají. Další detaily jsou zmíněny v podkapitole **4.4 (Příprava a prvotní nastavení aplikace)**

4.2 Příprava na práci s API Sceneform

Originální repozitář Googlu je na GitHubu ve stavu *Archived*, což znamená, že vývoj byl ukončen. Zatím nic nenasvědčuje, že by měl být obnoven, nebo že by šlo očekávat plnohodnotnou náhradu ze strany Googlu. Nicméně stále lze tuto knihovnu používat a při jejím testování jsem nenarazil na žádné problémy, které by to ztěžovaly. Nicméně je třeba vzít do úvahy rychlost vývoje Androidu, takže může kdykoliv nastat chvíle, kdy už tato verze Sceneformu zkrátka použitelná nebude.

Má aplikace používá *fork* originálního Sceneformu, o který se stará Thomas Gorisse. Oproti archivované verzi obsahuje nejnovější *backend* renderovacího API Filament a reflektuje aktuální trend používat AR modely a animace v .glb formátu, který lze načítat za běhu aplikace. Původní Sceneform stavěl na kombinaci *assetu* (.sfa) a binárního souboru (.sfb), které bylo třeba vytvořit pomocí pluginu do Android Studia a to pouze z .fbx nebo .obj souboru.

Tuto stále vyvíjenou verzi Sceneformu[3] můžeme snadno použít v nových projektech přidáním následujícího řádku do *dependencies* bloku v modulovém souboru build.gradle:

```
implementation("com.gorisse.thomas.sceneform:sceneform:1.18.9")
```

Takto snadno je to možné, protože vše potřebné je nahráno v Maven Central Repository, a v momentě, kdy chceme aplikaci sestavit, si Gradle všechny potřebné závislosti stáhne, pokud dosud staženy nebyly. Alternativně můžeme stáhnout celý repozitář nebo ho forknout z GitHubu a svou aplikaci vytvářet jako modul v tomto projektu, pak stačí odkazy na závislosti zkopírovat z ukázkových modulů, které projekt obsahuje (takto je vytvořena má aplikace).

[2]Google v dokumentaci uvádí, že pro úspěšnou detekci by měl tag zabírat alespoň 25% obrazovky telefonu [75]
[3]V době psaní tohoto textu je verze 1.18.9 nejnovější

4.3 Spolupráce na vývoji repozitáře

Jak už to tak bývá, programátoři občas nechají v kódu drobnou chybku, nedokonalost nebo na něco zapomenou. Ne vždy je to na první pohled vidět, ale občas nějaký bug vyjde na světlo. Do veřejného repozitáře na GitHubu je možné si napsat *issue*, což může být chápáno jako nahlášení chyby, ale rovněž se to dá použít k prostému položení dotazu nebo žádosti o pomoc. Člověk se pak zpravidla dočká odpovědi od některého z autorů repozitáře nebo aktivního člena komunity. Jednu menší chybu související se špatným načítáním textur u modelu, který jich obsahoval více různých jsem sám nahlásil [76] a pro její dobré zdokumentování jsem použil svou aplikaci. Autor repozitáře se podle všeho podrobněji podíval na mé zdrojové kódy a zaujala ho část, kde pracuji s detekcí tagů a jejich využití pro ukotvení 3D modelů, animací a videa. Poté mě požádal, jestli bych nepřipravil ukázkový kód, který by byl i ostatním veřejně k dispozici v jeho *forknutém* repozitáři. Souhlasil jsem a s hotovou ukázkou jsem vytvořil *pull request*, tedy žádost o přidání kódu uživatelem, který není vlastníkem repozitáře. Má ukázka se tak stala součástí původního repozitáře [77]. Navíc byla ona mnou původně nahlášená chyba ještě předtím velmi rychle opravena.

4.4 Příprava a prvotní nastavení aplikace

Už od úplného začátku vytváření projektu do Android Studia se musíme probrat spoustou parametrů a nastavení, které je nutné přesně určit. Definujeme tím vlastnosti zařízení, na které naše aplikace cílí, což např. v případě potřeby kompatibility s knihovnou ARCore jsou jen některé telefony a konkrétní verze systému. Verze Android API se ale mírně liší i v konstrukci jazyka a v dostupných GUI komponent.

4.4.1 Příprava emulátoru

Pro dle mého názoru nejpohodlnější testování aplikací je možné použít emulátor. Ten musí být postaven na Android API 27 a vyšším s architekturou x86_64 [20] a musí na něm být nainstalovaná speciální verze knihovny ARCore pro emulátor, kterou Google vždy vydává po boku té klasické. Jednak je tato speciální verze primárně určena pro architekturu x86_64[4] a jednak nejsme zatížení podporou ARCore pouze na určitých zařízeních. Průběh testování již je zmíněn v podkapitole **2.2.3 (Android emulátor)**.

4.4.2 arcoreimg tool

Tento nástroj je vytvořen Googlem aby co nejvíce usnadnil přípravné práce s AR tagy [41]. K dispozici je pro Windows, Linux a macOS ve formě souboru spustitelného z příkazového řádku

[4]Android emulátor s ARM architekturou není podporován, i když od verze Android API 30 lze spouštět ARM binárky na x86_64 zařízeních s vylepšenou efektivitou

(popř. terminalu), do kterého se vkládají argumenty. Ty určují, kterou ze dvou dostupných funkcí chceme použít, jaké soubory jsou na vstupu a kam uložit výstup.

4.4.2.1 Databáze tagů

První možnost využití je vytvoření z obrázků s příponou .png nebo .jpg jeden databázový soubor, který se poté v aplikaci pouze deserializuje a bez větších nároků na výkon se s jeho daty může začít pracovat. Databáze neobsahuje obrázky samotné, ale pouze z nich vytvořená data, která jsou založena na přechodech odstínů barev, liniích a dobře detekovatelných tvarech. Pokud známe skutečné rozměry tagu, lze je v metrech jako nepovinný údaj specifikovat a trošku tak zjednodušit jejich vyhledávání. Význam to ale začíná mít, až pokud je velikost tagu v řádu mnoha decimetrů nebo metrů. Výsledná databáze je binární .imgdb soubor s velikostí jednotek, maximálně desítek kilobajtů.

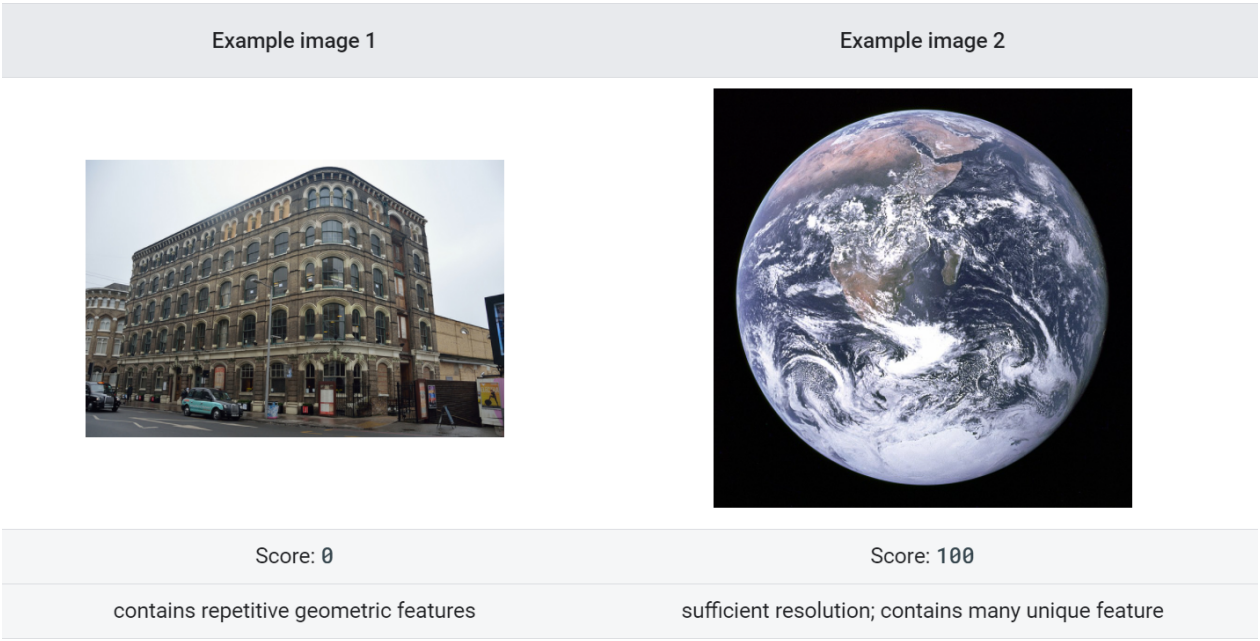
Pokud bychom chtěli obrázky načíst do aplikace přímo bez připravené databáze, tak i tento přístup je možný. Google ve své dokumentaci uvádí, že takovéto načtení jednoho obrázku zabere přibližně 30 milisekund [78]. O tom, jak k tomuto číslu došel a co ho nejvíce ovlivňuje, se ale nezmiňuje. Pokud bychom takto chtěli přidávat více obrázků, je vhodné si na to vytvořit nové vlákno, aby nedošlo k chvilkovému zpomalení nebo přímo zamrznutí aplikace kvůli zablokovanému UI vláknu. Předvytvoření databáze je tedy mnohem efektivnější a rychlejší, ale pokud bychom např. chtěli jako AR tag použít plakát, který jsme zrovna telefonem vyfotili, lze takto obrázek přidat až za běhu aplikace.

4.4.2.2 Ohodnocení kvality tagu

Druhá funkce dokáže vstupní obrázek bodově ohodnotit na stupnici 0 až 100, kde 100 je nejlepší výsledek, a tím nám dát představu o tom, jak dobře daný obrázek bude sloužit jako AR tag, viz obrázek **4.3 (Ukázka bodového ohodnocení špatně a dobře zvoleného AR tagu)**. Obecně se lze řídit zásadami, které jsou zmíněny i v dokumentaci [79]. Rozlišení obrázku by mělo být alespoň 300x300 pixelů, ale výrazně větší rozlišení zajistí jen nepatrné zlepšení chování. Velmi závisí na kompresi obrázku, které je nejlepší se zcela vyhnout. Co se týká vhodného vzhledu obrázku, tak není dobré, aby obsahoval velké množství špatně rozeznatelných detailů nebo častokrát se opakující tentýž vzor. Pro mou aplikaci jsem vytvořil sérii černobílých AR tagů s jednoduchými vzory a jeden s barevným textem, a to tak, aby dosáhly bodového ohodnocení 100, což se u všech povedlo.

4.4.3 Android Manifest

Aplikace využívající ARCore musí být určitým způsobem koncipovány, jednak aby byla zaručena jejich stoprocentní funkčnost, ale taky aby je Google mohl patřičně označit v případě publikace na GP. K tomuto účelu slouží soubor AndroidManifest.xml, tam se deklarují všechna povolení, která bychom chtěli pro aplikaci od systému získat [80]. Dále se uvádí názvy aktivit aplikace a která z nich



Obrázek 4.3: Ukázka bodového ohodnocení špatně a dobře zvoleného AR tagu, převzato z [79]

je hlavní. Systém je rovněž třeba takto upozornit na skutečnost, že implementujeme např. službu běžící na pozadí, broadcast receiver, atd. Zde je ukázka, jaká povolení jsou v Android Manifestu mé aplikace vyžadována:

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-feature android:name="android.hardware.camera.ar"/>
<uses-feature android:glEsVersion="0x00030000" android:required="true" />
```

Nejprve jedna žádost, konkrétně o povolení využívání kamery zařízení. Další dva řádky jsou oznamovací. První z nich říká, že aplikace vyžaduje podporu ARCore a druhý, že je potřeba na telefonu mít podporu OpenGL ES verze 3.0 a vyšší.

```
<meta-data android:name="com.google.ar.core" android:value="required" />
```

Tímto řádkem říkáme, že aplikace vyžaduje ARCore s hodnotou **required**, což znamená, že její obsah není možné bez podpory ARCore ani v omezené míře zobrazovat.

4.5 Ukázky kódu s vysvětlením

ARCore je knihovna postavená tak, aby se sama starala o obrazovou analýzu okolního prostředí spolu se zpracováním dat senzorů telefonu. Tím pádem je na programátorovi, aby zaručil, že knihovna ke všem potřebným datům bude mít přístup, aby si o její výstupní data řekl a správně je

použil. S API Sceneform se pracuje mnohem více, a proto i většina ukázkových kódů je zaměřena právě na něj, jeho třídy a vyžadované postupy. Kompletní zdrojový kód aplikace je k nalezení v příloze této práce, ale určitě není na škodu pro některé důležité části kódu přiložit vysvětlení, jaká je jejich funkce. *Při importování kousků kódů do tohoto textu jsem narazil na problémy s kódováním .java souborů, takže jsem musel u hlášek typu Toast odstranit diakritiku.*

4.5.1 Metoda onCreate

Metoda *onCreate* je jedna z hlavních pro práci s životním cyklem Android aktivit. Je zavolána mimo jiné ve chvíli, kdy je aplikace poprvé spuštěna. Využívám ji k prvotnímu nastavení vlastností vzhledu aplikace, inicializaci dat a nastavení potřebných listenerů. Následuje mírně upravená ukázka této metody převzaté z mé aplikace s vysvětlením funkce jednotlivých řádků, viz **4.1**.

Jelikož třída *MainActivity* dědí z *AppCompatActivity*, tak její metoda *onCreate* přepisuje metodu předka, musí tedy mít stejný indentifikátor viditelnosti a přijímat stejné parametry. Na řádku 3 se volá *onCreate* metoda předka, protože chceme chování při zapínání aplikace pouze rozšířit. Na řádku 5 je schována akční lišta aplikace a řádkem 6 se vynucuje pouze portrétové zobrazení, tedy na výšku. Hned poté je metodou *setContentView* spojena současná aktivita s XML souborem definující UI aplikace. Jelikož se pro zobrazování scény používá *FragmentContainerView*, můžeme u tohoto konkrétního typu fragmentu přidat listener na událost *onAttach*, která nastává jednou, když je fragment úspěšně vytvořen. Poté si ověříme, že je to skutečně námi očekávaný fragment podle jeho ID a když ano, uložíme jej jako proměnnou v rámci instance třídy *MainActivity*. Na řádcích 12 a 13 dáme překladači najevo, že budeme v rámci této třídy implementovat listener kliknutí na detekovanou rovinu *onTapPlane* a listener na inicializaci *session*, třídy z knihovny ARCore, *onSessionInitialization*. Můžeme se na tyto metody odkázat jen jako *MainActivity.this*, protože naše třída je značena tak, že implementuje *BaseArFragment.OnTapArPlaneListener* a *BaseArFragment.OnSessionInitializationListener*. Řádek 19 umožní promítat AR scénu živě do definovaného fragmentu, ale pouze v případě, že aplikace není spuštěna na pozadí (savedInstanceState bude *null*), a taky že zařízení splňuje náležitosti pro práci s knihovnou ARCore, potažmo Sceneform. Ve zbytku kódu se instancemi mé pomocné třídy *Model* naplní seznam, postupně se na každé jeho položek zavolá metoda *loadModel* a odpovídajícímu *ImageView* se nastaví listener na klik, který bude přepínat zrovna zvolený model.

```
1 @Override
2 protected void onCreate(Bundle savedInstanceState) {
3     super.onCreate(savedInstanceState);
4
5     getSupportActionBar().hide();
6     setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
7     setContentView(R.layout.activity_main);
8
```

```
9       getSupportFragmentManager().addFragmentOnAttachListener((fragmentManager,
            fragment) -> {
10          if (fragment.getId() == R.id.arFragment) {
11              this.arFragment = (ArFragment) fragment;
12              this.arFragment.setOnTapArPlaneListener(MainActivity.this);
13              this.arFragment.setOnSessionInitializationListener(MainActivity.
                    this);
14          }
15      });
16
17      if (savedInstanceState == null) {
18          if (Sceneform.isSupported(this)) {
19              getSupportFragmentManager().beginTransaction().add(R.id.arFragment
                    , ArFragment.class, null).commit();
20          }
21      }
22
23      modelList = new LinkedList<>();
24      modelList.add(new Model(ModelName.FOX, ModelUri.FOX, findViewById(R.id.
            foxView)));
25      modelList.add(new Model(ModelName.IRONMAN, ModelUri.IRONMAN, findViewById(
            R.id.ironmanView)));
26      modelList.add(new Model(ModelName.MERCEDES, ModelUri.MERCEDES,
            findViewById(R.id.mercedesView)));
27      modelList.add(new Model(ModelName.MUSHROOM, ModelUri.MUSHROOM,
            findViewById(R.id.mushroomView)));
28
29      for (Model model : modelList) {
30          loadModel(model);
31          model.getImageView().setOnClickListener(this);
32      }
33  }
```

Ukázka kódu 4.1: Metoda onCreate

4.5.2 Listenery

První listener *onTapPlane* již byl zmíněn v předchozí podkapitole a jeho obsah můžeme vidět zde: **4.2**. Je zavolán vždy, když uživatel klikne na již detekovanou rovinu. Nejprve se dvěma podmínkami ověří, že se aplikace nenachází v módu pro detekování AR tagů a že zvolený model je již načtený, aby nedocházelo k chybám. Na řádku 12 se vytvoří instance třídy *Anchor*, tedy kotva, která bude určovat přesný a neměnný bod na detekované rovině. Na základě kotvy se vytvoří *AnchorNode*, který určuje přesné souřadnice a rotaci ve scéně, ta se musí nastavit jako rodič. Na jeho základě je vytvořená

instance třídy *TransformableNode*, která se stará o práci s modelem, který se na řádku 18 nastaví a pokud obsahuje animaci, řádek 19 ji ihned spustí. V případě, že chceme model ve scéně někam posunout nebo otočit, vytvoříme si ještě *Node*, který bude mít rodiče typu *TransformableNode* a můžeme na něm zavolat metody *setLocalPosition* s parametrem typu *Vector3* pro posun nebo *setLocalRotation* s parametrem typu *Quaternion* pro rotaci.

```
1  @Override
2  public void onTapPlane(HitResult hitResult, Plane plane, MotionEvent
       motionEvent) {
3
4      if (inTagMode) {
5          Toast.makeText(this, "Pro umistovani modelu v prostoru prepnete mod
               aplikace", Toast.LENGTH_LONG).show();
6          return;
7      }
8      if (activeModel == null || activeModel.getRenderable() == null) {
9          return;
10     }
11
12     Anchor anchor = hitResult.createAnchor();
13     AnchorNode anchorNode = new AnchorNode(anchor);
14     anchorNode.setParent(arFragment.getArSceneView().getScene());
15
16     TransformableNode transNode = new TransformableNode(arFragment.
           getTransformationSystem());
17     transNode.setParent(anchorNode);
18     transNode.setRenderable(activeModel.getRenderable());
19     transNode.getRenderableInstance().animate(true).start();
20
21     Node node = new Node();
22     node.setParent(transNode);
23     node.setLocalPosition(new Vector3(0.0f, 1.0f, 0.0f));
24
25     activeModel.addAnchorNode(anchorNode);
26 }
```

Ukázka kódu 4.2: Listener onTapPlane

Druhý ukázkový listener je *onUpdate*, slouží pro detekci AR tagů a následné zobrazení modelu, či animace. Spuštěn je vždy v momentě, kdy je knihovnou ARCore zprostředkován další obrazový snímek se všemi dostupnými daty, které z něj bylo možné získat. Tato ukázka, viz **4.3**, byla pro účely demonstrace výrazně zkrácena na nezbytné funkční minimum jen pro jeden tag. Konktrétně zobrazí a spustí video, které přesně překryje detekovaný tag.

Řádek 2 zajistí ušetření výkonu procesoru ukončením, pokud se aplikace nenachází v módu hledání tagů nebo již všechny tagy z načtené databáze byly nalezeny. Na řádku 3 se ze scény získá aktuální snímek a z toho se na řádku 5 získají všechny AR tagy, které byly v současné scéně detekovány a jsou součástí databáze. AR tag zde představuje instanci třídy *AugmentedImage*. Déle se pak skrze jednotlivé tagy v kolekci iteruje a při každé iteraci se zjišťuje, jestli je tag ve stavu *TRACKING*, tedy je v tomto snímku v zorném úhlu kamery, jestli již náhodou nebyl detekován, a nakonec jestli jeho identifikátor je shodný s *"matrix.png"*. Pokud je vše splněno, tak je třeba video umístit a spustit.

Většina tohoto kódu je společná s metodou pro zobrazování modelů zmíněná výše, viz **4.2**. Model představující projekční plátno již je od prvotního spuštění aplikace načtený v proměnné *plainVideoModel*, proto je možné jej rovnou použít, to stejné platí pro jeho texturu *plainVideoMaterial*. Velmi důležitý je řádek 13, který přizpůsobuje velikost *AnchorNode* na velikost detekovaného tagu a zajistí jeho kompletní překrytí. Další zajímavý je řádek 18, který ukazuje použití rotace modelu podle quaternionu. Video se z neznámého důvodu obrazuje vzhůru nohama, a proto je třeba jej takto otočit. Další novinka je využití třídy *MediaPlayer*, u které se nastaví zdroj videa, jestli se má přehrávat pořád dokola, povrch projekčního plátna, a poté se přehrávání spustí.

```
1  public void onUpdate(FrameTime frameTime) {
2      if (!inTagMode || !anyUntrackedTagExists()) return;
3      Frame frame = this.arFragment.getArSceneView().getArFrame();
4      try {
5          Collection<AugmentedImage> augmentedImageCollection = frame.
               getUpdatedTrackables(AugmentedImage.class);
6          for (AugmentedImage image : augmentedImageCollection) {
7              if (image.getTrackingState() == TrackingState.TRACKING) {
8                  if (!tagsDetected.get(image.getName()) && image.getName().
                       equals("matrix.png")) {
9                      tagsDetected.replace(image.getName(), true);
10                     findViewById(R.id.tagSquare).setVisibility(View.GONE);
11
12                     AnchorNode anchorNode = new AnchorNode(image.createAnchor(
                           image.getCenterPose()));
13                     anchorNode.setWorldScale(new Vector3(image.getExtentX(), 1
                           f, image.getExtentZ()));
14                     arFragment.getArSceneView().getScene().addChild(anchorNode
                           );
15
16                     TransformableNode videoNode = new TransformableNode(
                           arFragment.getTransformationSystem());
17                     videoNode.setParent(anchorNode);
```

```
18                      videoNode.setLocalRotation(Quaternion.axisAngle(new
                            Vector3(0, 1f, 0), 180f));
19                      videoNode.setRenderable(this.plainVideoModel);
20
21                      ExternalTexture externalTexture = new ExternalTexture();
22                      RenderableInstance renderableInstance = videoNode.
                            getRenderableInstance();
23                      renderableInstance.setMaterial(this.plainVideoMaterial);
24                      renderableInstance.getMaterial().setExternalTexture("
                            videoTexture", externalTexture);
25
26                      MediaPlayer mediaPlayer = MediaPlayer.create(this, R.raw.
                            matrix);
27                      mediaPlayer.setLooping(true);
28                      mediaPlayer.setSurface(externalTexture.getSurface());
29                      mediaPlayer.start();
30                  }
31              }
32          }
33      }
34      catch (Exception e) {
35          e.printStackTrace();
36      }
37  }
```

Ukázka kódu 4.3: Listener onUpdate

4.5.3 Práce s modely

Následuje ukázka kódu **4.4** metody *loadModel*, která je volána v metodě *onCreate* pro každou vytvořenou instaci mé pomocné třídy *Model*. Na začátku se vytvoří *WeakReference*, která bude představovat slabou referenci na již existující instanci třídy *MainActivity*, kterou v tomto kontextu reprezentuje klíčové slovo *this*. Slabá reference dovoluje *garbage collectoru* onu referenci z paměti odstranit, když už na ni nevede žádná silná reference (tedy známá reference pomocí klíčového slova *new*). Tento postup je zvolen proto, že životní cyklus aktivit na Androidu v mnoha případech dovoluje, aby byla instance třídy *MainActivity* vytvořena znovu a zároveň načítání souboru s modelem je asynchronní a zabere nějaký čas. Kdyby byla použita silná reference, byla by stále platná i když by byla hlavní aktivita vytvořena znova. Použitím slabé reference lze zabránit možnému *memory leaku*.

Dále od řádku 3 se využívá metoda *builder*, která vytvoří instanci třídy *ModelRenderable*. Nastaví se cesta k souboru. Dále že se jedná o soubor typu .glTF, kompatibilní s Filamentovým API (.glb je binární verze .glTF), a model se ze souboru sestaví pomocí *build()*. Sestavení je ale asynchronní

proces, který vrací objekt typu *CompletableFuture<ModelRenderable>*, takže nastavíme listener na lambda výraz, který se spustí až bude sestavení hotovo. Zároveň nastavíme i listener, který se zavolá v případě výjimky. Na řádku 8 a 9 si ověříme, že slabá reference je stále platná, a pokud je, na řádku 10 se načtený a sestavený model předá instanci třídy *Model*. Poté řádky 11 a 12 deaktivují zobrazení stínů u načteného modelu a následný podmíněný blok nastavuje prvotní stav aplikace, aby hned po spuštění byl z nabídky vybrán model lišky a bylo jej možno hned použít.

```
1  public void loadModel(Model modelInstance) {
2      WeakReference<MainActivity> weakActivity = new WeakReference<>(this);
3      ModelRenderable.builder()
4          .setSource(this, Uri.parse(modelInstance.getUri()))
5          .setIsFilamentGltf(true)
6          .build()
7          .thenAccept(model -> {
8              MainActivity activity = weakActivity.get();
9              if (activity != null) {
10                 modelInstance.setRenderable(model);
11                 model.setShadowCaster(false);
12                 model.setShadowReceiver(false);
13                 if (modelInstance.getName().equals(ModelName.FOX)) {
14                     activity.activeModel = modelInstance;
15                     modelInstance.getImageView().setBackgroundColor(Color.
                           parseColor("#99000000"));
16                 }
17             }
18         })
19         .exceptionally(throwable -> {
20             Toast.makeText(this, "Nelze nacist model " + modelInstance.getUri
                   (), Toast.LENGTH_LONG).show();
21             return null;
22         });
23 }
```

Ukázka kódu 4.4: Načtení modelu

4.6 Debugging

Pochopitelně jako u každého většího projektu je nutné výsledek dobře otestovat. Pokud se chystáme na zveřejnění aplikace pro větší skupinu lidí s různými telefony, můžeme očekávat i různé zpětné vazby k rychlosti a funkčnosti. Výhoda emulátoru určitě spočívá v jeho jednoduché upravitelnosti a možnosti snadno změnit virtuální zařízení nebo verzi Androidu. I přes velikou přesnost emulace to ale není opravdový telefon a měli bychom s tím počítat.

Když jsem potřeboval novou verzi aplikace vyzkoušet, tak první pokusy byly vždy v emulátoru. Má zkušenost je taková, že ve většině případů je emulátor méně výkonný a méně plynulý než telefon, ale pokud je nějaká funkce v kódu velmi neefektivní a náročná, tak se to paradoxně významně projeví teprve až na reálném zařízení, kde aplikace třeba i spadne. Největší výkonnostní rozdíl je určitě v práci se soubory a v případě mé aplikace jde hlavně o jejich načítání. I když většina souborů s modely nezabírá více než 10 megabajtů, tak jejich načítání se na plynulosti běhu aplikace projeví.

Prostředí VirtualScene pro testování rozšířené reality je velmi dobře vytvořené. Když je aktivní okno emulátoru, tak podržením klávesy Alt se lze pohybem myši v místnosti otáčet a v kombinaci s klávesnicí pohybovat. Používají se hráčům dobře známé klávesy WASD pro pohyb v horizontální rovině a klávesy Q a E pro pohyb nahoru a dolů. Teprve až když jsem už nenacházel žádné chyby v emulátoru, tak jsem pro testování sáhl po skutečném telefonu a začal se zaměřovat na uživatelskou přívětivost.

4.7 Vizuální obsah aplikace

Z hlediska prostého uživatele je vizuální obsah této aplikace to jediné zajímavé, a proto by měl maximálně poutat jeho pozornost. Obecně jsem se ve svém okolí nejčastěji setkával s neznalostí rozšířené reality a tím pádem i v jednoduchém podání vyvolává nadšení a zájem, už jen proto, že ji na svém telefonu může vyzkoušet většina lidí sama. Dosavadní obsah aplikace záměrně zcela neodpovídá budoucímu obsahu pro účely použití na hradě Sovinec, protože tato aplikace byla vytvořena především pro účely demonstrace a hlubší prozkoumání možností rozšířené reality. Zároveň řešení problémů, na které jsem doposud narazil, mi pomůže při budoucím rozšiřování aplikace o komplexnější obsah hlavně ve fromě animací.

4.7.1 Použité modely

Aplikace obsahuje mnoho modelů k zobrazení, a to buď přímo na detekované rovině, nebo skenováním AR tagu. Některé byly převzaty z veřejně dostupných zdrojů, jako model lišky [19], IronMana [81] (ten byl stažen již v roce 2014 a byl tehdy k dispozici zcela zdarma) a pro další úpravy byla zvolena animace chůze člověka, která vznikla profesionálním skenováním technologií MoCap [82] s volnou licencí i pro komerční použití.

Modely představující automobil Mercedes AMG GT, muchomůrky a králíka jsou má vlastní, ale již dřívější tvorba. Animaci otevírání trezoru jsem vytvořil až pro tuto aplikaci. Video představující vzhled konzole jako z filmu Matrix bylo převzato z YouTube [83]. Model vojáka s halapartnou (viz obrázek **4.4**) obsahující samostatně vytvořené animace byl zakoupen a tyto animace byly spojeny v jednu.

Obrázek 4.4: Zakoupený animovaný model halapartníka s přidanou helmou [84]

4.7.2 Zakoupené modely

Dělat složité animace úplně od základu by pro mne bylo časově velmi náročné, jelikož musím přiznat, že jsem v tomto ohledu stále začátečník. To platí hlavně v případě, že by se mělo jednat o kompletní tvorbu modelu člověka a přidání rigu s následným animováním např. jeho chůze. Navíc by výsledek u složitějších animací zcela jistě moc věrohodně nepůsobil a tím pádem by klesala kvalita výsledné aplikace. Pro použití na hradě Sovinec jsme se tedy domluvili, že bude nejlepší zakoupit profesionálně vytvořené modely postav, které navíc obsahují rig, a dokonce i ukázky animací pohybu. Takovéto modely mohou velmi dobře posloužit jako výchozí stav pro další úpravy, taky proto, že podmínky jejich použití zahrnují i publikaci komerčního charakteru.

4.7.3 Úprava existujících animací

Aby zakoupené animace odpovídaly požadavkům, co se týká jejich obsahu, je třeba je upravit. Toho lze nejlépe dosáhnout nejlépe pomocí editace klíčových snímků. Je možné, že již existující animace bude alespoň částečně využitelná, ale ani tak se stoprocentním uplatněním nelze počítat. Při práci s klíčovými snímky se mi nejvíce osvědčil postup, při kterém si u animace určím klíčové snímky začátku a konce a postupně mezi ně vkládám další snímky upřesňující animaci. Takto animační prostředí programu vytváří plynulý přechod mezi snímky a toho je dle mne velmi dobré využívat. Při postupném přidávání klíčových snímků jednoho po druhém od začátku až po konec animace

se mi nedařilo docílit plynulé návaznosti jednotlivých klíčových snímků a bylo nutné je vícekrát upravovat a posouvat, což velmi zdržuje.

Pokud by bylo cílem vytvořit animaci přikývnutí hlavou na postavě s již vytvořeným rigem, ale prozatím bez jakýchkoliv existujících klíčových snímků, postup by mohl vypadat následovně:

1. Určíme si celkovou délku animace. Pro ilustraci můžeme zvolit interval jedné sekundy, při snímkovací frekvenci 30FPS máme tedy k dispozici rozpětí třiceti snímků.

2. Vybereme krční kloub rigu a nastavíme mu současnou rotaci jako klíčový snímek první a třicátý snímek na časové ose. Chceme totiž začít i skončit animaci se stejnou pozicí hlavy.

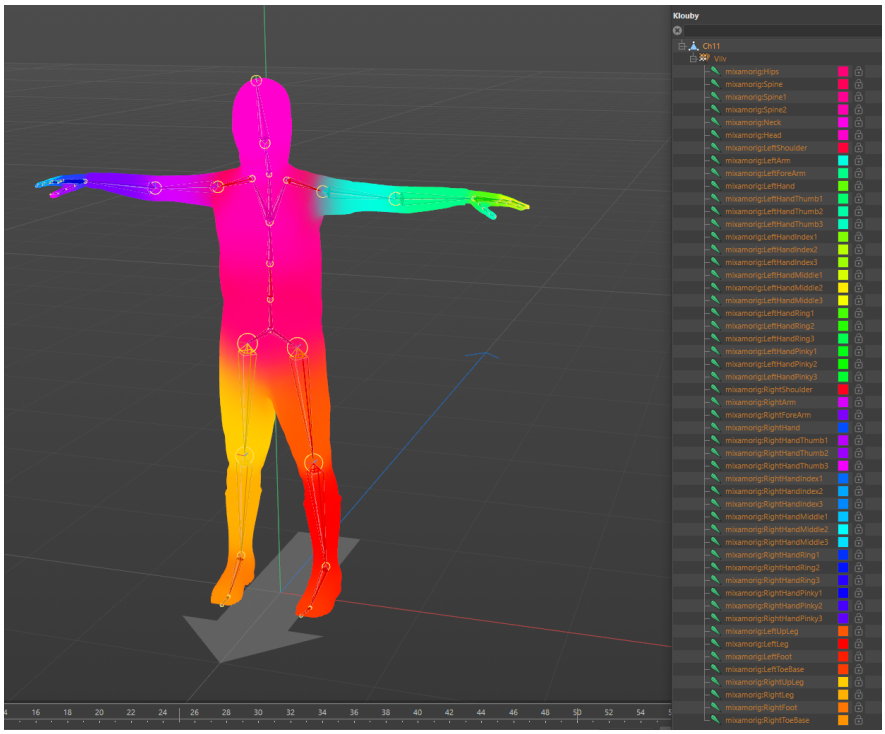
3. Přejdeme na snímek 14 a ohneme krční kloub do požadované míry, pochopitelně v rámci možností lidského těla. Pro tento ohyb vytvoříme klíčový snímek a aby se v této pozici hlava na chvíli zastavila, vytvoříme úplně stejný klíčový snímek i pro následující snímek animace (15).

4. Aby pohyb hlavou vypadal méně strojově, můžeme mezi současné klíčové snímky přidat další, které by způsobily mírné naklonění hlavy na stranu nebo nějakou jinou nepravidelnost např. upravující rychlost animace.

5. Základ animace je hotový a můžeme ji spustit. Pokud se nám nelíbí rychlost animace, můžeme klíčové snímky na časové ose posouvat nebo animaci doladit přidáním dalších mezi již existující.

Animace je možné za určitých podmínek spojovat, a to konkrétně za předpokladu, že vycházíme ze dvou různých projektů, kde ale každý pracuje se stejným modelem, který obsahuje úplně stejný rig. Zakoupené modely jsou vetšinou udělány tak, že pokud obsahují více různých animací, tak ty jsou vytvořené jako samostatné soubory. Nechceme ale kopírovat celou scénu i s modelem, nýbrž chceme pouze klíčové snímky jedné animace aplikovat na rig té druhé. K tomu v prostředí Cinemy slouží tzv. klipy pohybu a jejich editace v časové ose.

Do projektu musíme nejprve vložit všechny animace, které chceme spojit, a pro lepší organizaci si každou zvlášť obalíme objektem Osy. Pro každou animaci, kterou představují jednotlivé Osy, přidáme klip pohybu. Z animací vybereme tu, kterou chceme aby po spojení výsledná animace začínala a zvolíme, že chceme klip pohybu zobrazit v časové ose. Do té pak máme možnost vložit i ostatní klipy pohybu, libovolně si je uspořádat, a třeba i prokládat (pak se v místě překrytí jejich pohyby zprůměrují). Jestliže máme časovou osu uspořádanou, můžeme z projektu vymazat všechny objekty, kromě těch, na kterých časová osa závisí, tedy ty, které představovaly první animaci. Animaci máme sice spojenou, ale ztratíme přístup ke klíčovým snímkům.



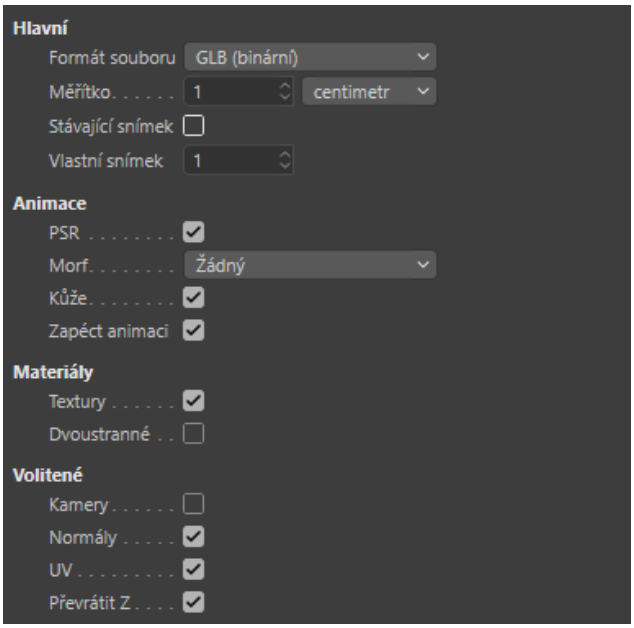
Obrázek 4.5: Rozvržení vlivu kloubů rigu na kůži postavy

4.7.4 Změna oblečení postav

Změnu oblečení postav bezesporu považuji za naprosto nejsložitější a nejzdlouhavější proces, který jsem v rámci této aplikace dělal. Tímto procesem ale není myšlena změna textury kůže, nýbrž doslova navlečení jednoho nebo více polygonových objektů představujících oblečení na již animovanou postavu. Kromě nulových předchozích zkušeností k tomu napomáhá i problematický export animací (viz **4.8**). Existuje celá řada tutoriálů na tvorbu oblečení animovaných postav v programu Cinema4D, ukazující velmi pokročilé funkce tohoto programu, které jsou zároveň velmi snadné na použití. Problém je ten, že export takovýchto funkcí do formátu .glb není podporován a jediný způsob oblečení postavy, který se mi úspěšně podařilo exportovat je využití mapování vlivu kostry na polygonový objekt oblečení.

Přidáním tagu *Vliv* na libovolný polygonový objekt představující oblečení lze nastavovat, které konkrétní existující klouby by na oblečení měly mít vliv. Výsledný posun oblečení a jeho deformace v čase by pak měla odpovídat pohybu postavy. Upravit lze míru ovlivnění výsledku pro jednotlivé klouby a také počet průchodů při výpočtu během vytváření animace, což ve výsledku lépe vyhladí tvar oblečení, ale trvá déle. Při editování můžeme vidět, který kloub rigu má největší vliv na danou část oblečení, což je ukázáno na obrázku **4.5**. Tam je tímto způsobem utvářen pohyb polygonové kůže postavy, oblečení navíc sice tento model nemá, ale princip je úplně stejný.

Druhý exportovatelný způsob přidávání oblečení je mnohem jednodušší a neumožňuje oblečení



Obrázek 4.6: Možnosti exportu Cinema4D projektu do .glb formátu

dle potřeby deformovat. Využívá funkci *Omezení PSM*, která pouze sleduje pohyb jednoho konkrétního kloubu a udržuje od něj pořád stejnou vzdálenost i relativní pozici. Tuto funkci jsem použil, když jsem potřeboval halapartníkovi nasadit přilbu, která by se korektně pohybovala s jeho hlavou, viz **4.4**. Je ale důležité, aby ve výčtu všech objektů projektu v prostředí Cinemy byl objekt s omezením umístěn až po rigu, na kterém je závislý, jinak by byl tento objekt ve svém pohybu vždy o jeden snímek pozadu oproti animaci rigu a tedy i postavy.

4.8 Export scén

Cinema4D umožňuje export projektů do velikého množství různých formátů, ale pro účely AR je podstatný export hlavně do .glb formátu. Správné nastavení takového exportu mi bohužel zabralo až příliš mnoho času. Konkrétní nastavení exportu lze vidět na obrázku **4.6**. Toto dialogové okno neobsahuje žádné nápovědy a ani popisky, jaký vliv bude nastavení mít na export a je tedy třeba mít v této oblasti buď veliký přehled, všechno dohledávat nebo prostě zkoušet.

Další nepříjemné zjištění bylo, že naprostou většinu svých pokročilejších funkcí Cinema neumí převést na primitivnější třeba použitím klíčových snímů pro každý bod scény zvlášť. Byl jsem tedy zcela odkázán na nalezený seznam podporovaných funkcí, vlastností a nastavení textur, který Maxon vydal [85] pro původně samostatný plugin. Během posledního více než roku byla ale jediná větší změna tohoto pluginu, že už se stal přímo součástí Cinemy.

Problémové je i měřítko objektů. Závislost mezi měřítkem původního projektu a toho exportovaného je mi stále záhadou. Při úplně prvním pokusu o umístění mnou vytvořené animace do

rozšířené reality se nikde v prostoru nezobrazovala a dlouho jsem nechápal proč. Nakonec jsem zjistil, že model, který v prostředí Cinemy má výšku průměrného člověka se exportem zvětšil na výšku několika stovek metrů a ARCore si s něčím tak velkým neví rady. Takovéto nedorozumění lze kompenzovat řádovým zvětšením jednotky měřítka v nastavení exportu nebo zmenšením výchozího modelu samotného do řádu desetin centimetru. Podobný problém je i ve vztahu emulátoru a reálného zařízení. Pokud už se při testování v emulátoru zdálo, že je velikost přijatelná, vyplatilo se model ještě zmenšit zhruba na polovinu a teprve potom to zkusit i na reálném telefonu. Těžko se sice odhaduje velikost virtuální místnosti, ale osobně mám pocit, že tam měřítko taky moc nesedí.

4.9 Textury a jejich vzhled v reálu

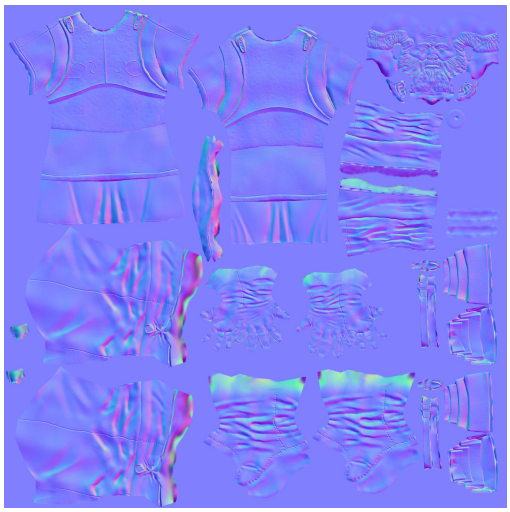
Textury představovaly další problém, který stále nemá z mé strany úplně jednoznačné řešení. Nejen, že export do .glb formátu nepodporuje většinu možných nastavení a vlastností materiálů, ale i když jsem s tím počítal podle dostupné dokumentace, tak samotný vzhled materiálů se velmi lišil. Materiály měly na modelech úplně jiné vlastnosti, pokud byly zobrazeny ve 3D Prohlížeči Windows, online prohlížeči nebo v aplikaci, zřejmě si podání barev každý implementuje jinak. Nepřišel jsem na lepší metodu než každou malou změnu hned aplikovat a otestovat v aplikaci bez ohledu na to, jak vypadá v 3D editorech. Naštěstí se vzhled textur nelišil mezi emulátorem a reálným zařízením.

Pro zkušené designéry to ovšem nemusí představovat problém, protože načítání textur z obrázku funguje velmi konzistentně. U detailních modelů především lidí a zvířat nebo při napodobení struktury nějakého materiálu (cihly, beton, dřevo, ...) se kompletní obalová textura rozloží do jednoho obrázku a z ní je potom pomocí UV mapování přidělena jednotlivým polygonům objektu. Takto vytvářet textury bezesporu vyžaduje umělecké dovednosti, pokud by výsledek měl být hezký, protože texturu je třeba opravdu nakreslit. Další zajímavé zpestření textur je přidání jim iluzi hloubky, což mohou být záhyby na látce, rýhy ve skle nebo třeba letokruhy dřeva. Toho lze velmi věrně docílit použitím normál, které se stejně jako textury z externího obrázku namapují na správné polygony. Bohužel mám s tvorbou takovýchto textur jen velmi málo zkušeností, takže jsem se do tvorby tímto způsobem nepouštěl, nicméně u koupených modelů je výsledného vzhledu dosaženo právě kombinací UV mapování textur (viz obrázek **4.7**) a normálové vrstvy (viz obrázek **4.8**).

4.10 Výsledná aplikace

Celkově aplikace po instalaci na reálné zařízení zabírá necelých 70 MB, což není tak moc, když průměr velikosti aplikací např. v mém telefonu je okolo 100 MB. Navíc se aktivně nevyužívá paměť zařízení, takže se tato hodnota nebude měnit v průběhu používání. Pro funkčnost na telefonech je vyžadována přítomnost podpůrné služby ARCore v současně nejnovější verzi, tento požadavek na minimální verzi se ale měnit nebude. Minimální verzi Android API jsem zvolil na 27, tedy Android ve verzi 8.1 s označením Oreo, což podle dat z Android Studia by mělo zajistit funkčnost na cca 50

Obrázek 4.7: Ukázka souboru s texturami pro UV mapování na objekt



Obrázek 4.8: Ukázka souboru textury normálové vrstvy

% zařízeních, viz **2.2 (Procentuální zastoupení verzí Androidu mezi koncovými uživateli k lednu 2021)**.

Protože v rámci této práce bylo třeba co nejlépe demonstrovat možnosti rozšířené reality na Androidu a tím pádem implementovat co možná nejvíce různých funkcí, současná aplikace se od té budoucí, která bude použita na Sovinci, mírně liší. Je vytvořená tak, aby její budoucí změny z hlediska kódu byly zcela minimální a spíše záležitost kopírování nebo mírné úpravy toho stávajícího. Další vývoj bude zaměřen převážně už jen na tvorbu vizuálního obsahu ve formě animací. Co se jich týká, tak očekávám, že budou zabírat určitě více místa než teď, a proto bych do konečné verze implementoval načtení všech modelů již při startu aplikace, ale ve vláknech, aby nebylo zbytečně vytěžováno UI vlákno a zrychlila se odezva při detekování tagu. V současném stavu to nepředstavuje až takový problém na zařízeních, které jsem měl pro test k dispozici, a celková plynulost se mi zdá dobrá, ale paralelismus představuje prostor pro zlepšení, kterého bych chtěl využít.

V obou módech aplikace se vyžaduje vcelku minimální interakce uživatele. V prvním stačí kliknout na nabídku pro vybrání modelu, a pak kliknout na detekovanou rovinu, jejichž detekce probíhá automaticky pohybem zařízení v prostoru. A ve druhém módu je třeba zařízení přiblížit k AR tagu, jehož detekce probíhá rovněž bez zásahu uživatele. Ten je tedy primárně staven do role diváka, a ne tvůrce děje.

V rámci testování jsem měl k dispozici jednak emulátor na desktopovém OS Windows a také 2 telefony. U telefonů se jednalo o Samsung Galaxy M21 s Androidem 10 a později po aktualizaci s Androidem 11 a druhý byl Xiaomi Redmi Note 9S s Androidem 10. Co se týká emulátoru, tak jsem použil přednastavený typ Pixel 4 s Android API 29 a architekturou x86_64 (na podpoře Google Play nezáleží) u kterého jsem si zvýšil velikost RAM paměti z 1536MB na 4096MB a ověřil, že je u zadní kamery nastaven VirtualScene, ostatní nastavení jsou ponechány na výchozích hodnotách.

Bohužel obecně známý bug Android emulátoru způsobuje, že na Linuxových systémech testování AR aplikací nemusí fungovat. Nejspíše je to způsobeno implementací nebo nastavením OpenGL. Ve výsledku se aplikace sice spustí, ale je vidět jen její GUI na černém pozadí. Pro testování v emulátoru tedy doporučuji OS Windows.

Kapitola 5

Závěr

Výstupem této práce je funkční aplikace v rozšířené realitě, kterou si lze nainstalovat na mobilní telefon se systémem Android, jež je podporován knihovnou ARCore. Aplikace bude obsahově dále rozšiřována, aby posléze mohla sloužit návštěvníkům na hradě Sovinec k ozvláštnění prohlídek. Bohužel dodání podkladů ze strany Sovince a jejich dodavatelů bylo poznamenáno pandemickou krizí, což pozdrželo i práce na aplikaci. Proto bude vývoj převážně vizuálního obsahu pokračovat i po publikaci této práce.

Před samotnou tvorbou aplikace jsem se teoreticky seznámil s mnoha knihovnami nejen na operační systém Android, které se rozšířenou realitou zabývají. Nakonec jsem zvolil knihovnu ARCore, která je vyvíjena Googlem a jejíž popularita na poslední dobu velmi rychle roste. Jako renderovací API jsem zvolil Sceneform z repozitáře nezávislého vývojáře na GitHubu, do kterého jsem malým dílem přispěl i já a částečně tak snad pomohl v dalším rozvoji tohoto API. Díky tomu jsem více pochopil, jakým způsobem jsou provozovány veřejně dostupné repozitáře a jak do jejich vývoje může být zapojen jakýkoliv člen komunity.

Musel jsem se rovněž naučit, jakým způsobem vytvářet obsah vhodný pro použití v rozšířené realitě hlavně s ohledem na stejnorodost vzhledu v animačním prostředí a na telefonu. Stejně tak je ale třeba velmi šetřit místem a výpočetním výkonem potřebným k zobrazení takového obsahu na mobilním zařízení. V této oblasti vidím největší prostor ke zlepšení, protože vhledem ke špatné kompatibilitě exportu projektu do požadovaného souborového formátu je možné používat jen základní funkce, takže práce je pak velmi zdlouhavá s nejistým výsledkem. Za obrovskou škodu považuji, že jsem nemohl naplno používat tak veliký potenciál, jaký mnou zvolený 3D animační software Cinema4D nabízí. To by práci jistě usnadnilo.

Tato práce byla celkově dle mého názoru velmi široce zaměřena, což mi jednak poskytlo volnější téma ke zpracování, ale hlavně jsem mohl prohlubovat své znalosti a schopnosti ve více různých odvětvích najednou. Sice jsem v jejím průběhu narážel na značné množství nejrůznějších problémů, ale na druhou stranu se je postupně dařilo vcelku eliminovávat a během toho jsem se spoustu nového naučil. S výsledkem práce jsem spokojen.

Literatura

1. *IKEA Place - AR app* [online] [cit. 2021-03-27]. Dostupné z: https://www.ikea.com/au/en/customer-service/mobile-apps/say-hej-to-ikea-place-pub1f8af050.
2. *The Witcher - Monster Slayer* [online] [cit. 2021-03-27]. Dostupné z: https://www.press.bmwgroup.com/global/article/detail/T0294345EN/absolutely-real:-virtual-and-augmented-reality-open-new-avenues-in-the-bmw-group-production-system?language=en.
3. *Pokémon GO* [online] [cit. 2021-03-27]. Dostupné z: https://play.google.com/store/apps/details?id=com.nianticlabs.pokemongo.
4. *The Witcher - Monster Slayer* [online] [cit. 2021-03-27]. Dostupné z: https://thewitcher.com/cz/en/monster-slayer.
5. *9 Powerful Real-World Applications Of Augmented Reality (AR) Today* [online]. 499 Washington Blvd., New Jersey: Forbes, Inc., 2018 [cit. 2021-03-27]. Dostupné z: https://blogs-images.forbes.com/bernardmarr/files/2018/07/AdobeStock_124464399.jpeg.
6. *Hello, Android : introducing Google's mobile development platform*. 3rd ed. Raleigh, N.C.: The Pragmatic Bookshelf, 2010. ISBN 978-1-93435-656-2.
7. *Developer Guides* [online] [cit. 2021-04-19]. Dostupné z: https://developer.android.com/guide.
8. *How JVM Works – JVM Architecture?* [Online] [cit. 2021-04-19]. Dostupné z: https://www.geeksforgeeks.org/jvm-works-jvm-architecture/.
9. *Google (Alphabet) vyhrál spor s Oracle o Android* [online] [cit. 2021-04-19]. Dostupné z: https://www.root.cz/zpravicky/google-alphabet-vyhral-spor-s-oracle-o-android/.
10. *GO* [online] [cit. 2021-04-19]. Dostupné z: https://golang.org/.
11. *Paint your UI to life* [online] [cit. 2021-04-19]. Dostupné z: https://dart.dev/.
12. *Xamarin* [online] [cit. 2021-04-19]. Dostupné z: https://dotnet.microsoft.com/apps/xamarin.

13. *NuGet* [online] [cit. 2021-04-19]. Dostupné z: https://www.nuget.org/packages.

14. *Design beautiful apps* [online] [cit. 2021-04-19]. Dostupné z: https://flutter.dev/.

15. *React Native - Learn once, write anywhere.* [Online] [cit. 2021-03-29]. Dostupné z: https://reactnative.dev/.

16. *Meet Android Studio* [online]. Google, [b.r.] [cit. 2021-03-27]. Dostupné z: https://developer.android.com/studio/intro.

17. *Gradle build tool* [online]. Gradle, [b.r.] [cit. 2021-03-31]. Dostupné z: https://gradle.org/.

18. *Android Debug Bridge (adb)* [online]. Google, [b.r.] [cit. 2021-03-27]. Dostupné z: https://developer.android.com/studio/command-line/adb.

19. BLAKELEY, Jake. *Low-poly fox* [online]. 2019 [cit. 2021-03-27]. Dostupné z: https://poly.google.com/view/9C-MLNfxaor.

20. *Run AR Apps in Android Emulator* [online]. Google, [b.r.] [cit. 2021-03-27]. Dostupné z: https://developers.google.com/ar/develop/java/emulator.

21. *Dive into the world of augmented reality.* [Online]. Apple, [b.r.] [cit. 2021-04-01]. Dostupné z: https://developer.apple.com/augmented-reality/.

22. *3D Object Recognition with ARKit 2* [online] [cit. 2021-04-02]. Dostupné z: https://www.youtube.com/watch?v=wrYYN3a6XZo.

23. ŘEHÁČEK, Ondřej. *Rozšířená realita* [online]. Ostrava, 2019 [cit. 2021-04-02]. Dostupné z: http://hdl.handle.net/10084/136119. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava.

24. *About AR Foundation* [online]. Unity, [b.r.] [cit. 2021-04-03]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.1/manual/.

25. *Vuforia: Market-Leading Enterprise AR* [online] [cit. 2021-04-03]. Dostupné z: https://www.ptc.com/en/products/vuforia.

26. *Vuforia Engine Pricing: The Right Plan to Build Your Vision* [online] [cit. 2021-04-03]. Dostupné z: https://www.ptc.com/en/products/vuforia/vuforia-engine/pricing.

27. *Modernize training and work instructions with Vuforia Studio* [online] [cit. 2021-04-03]. Dostupné z: https://forums.macrumors.com/threads/photo-face-id-infrared-dot-matrix.2084242/.

28. *AUGMENTED REALITY* [online] [cit. 2021-04-03]. Dostupné z: https://www.wikitude.com/.

29. *Download Wikitude SDK for cross-platform AR experiences* [online] [cit. 2021-04-03]. Dostupné z: https://www.wikitude.com/download/.

30. *Wikitude Studio* [online] [cit. 2021-04-03]. Dostupné z: https://www.wikitude.com/products/studio/.

31. *artoolkitX* [online] [cit. 2021-04-02]. Dostupné z: http://www.artoolkitx.org/.

32. *Augmented Reality SDK* [online] [cit. 2021-04-02]. Dostupné z: https://www.deepar.ai/.

33. *EasyAR* [online] [cit. 2021-04-03]. Dostupné z: https://www.easyar.com/view/sdk.html.

34. *ARCore overview* [online]. Google, [b.r.] [cit. 2021-04-02]. Dostupné z: https://developers.google.com/ar/discover.

35. *Depth API quickstart for Android* [online]. Google, [b.r.] [cit. 2021-04-03]. Dostupné z: https://developers.google.com/ar/develop/java/depth/quickstart.

36. *Fundamental concepts* [online]. Google, [b.r.] [cit. 2021-04-02]. Dostupné z: https://developers.google.com/ar/discover/concepts.

37. *Instant Placement overview for Android* [online]. Google, [b.r.] [cit. 2021-03-27]. Dostupné z: https://developers.google.com/ar/develop/java/instant-placement/overview.

38. *Quickstart for Android* [online]. Google, [b.r.] [cit. 2021-03-27]. Dostupné z: https://developers.google.com/ar/develop/java/quickstart.

39. *Depth API overview for Android* [online]. Google, [b.r.] [cit. 2021-03-27]. Dostupné z: https://developers.google.com/ar/develop/java/depth/overview.

40. *Augmented Images for Android* [online]. Google, [b.r.] [cit. 2021-03-27]. Dostupné z: https://developers.google.com/ar/develop/java/augmented-images.

41. *The arcoreimg tool* [online]. Google, [b.r.] [cit. 2021-03-27]. Dostupné z: https://developers.google.com/ar/develop/java/augmented-images/arcoreimg.

42. *OpenGL* [online]. Khronos group, [b.r.] [cit. 2021-04-19]. Dostupné z: https://www.khronos.org/opengl/wiki/Getting_Started.

43. *OpenGL ES* [online]. Google, [b.r.] [cit. 2021-04-03]. Dostupné z: https://developer.android.com/guide/topics/graphics/opengl.

44. *Vulkan* [online]. Khronos group, [b.r.] [cit. 2021-04-19]. Dostupné z: https://www.khronos.org/vulkan/.

45. *Filament* [online]. Google, [b.r.] [cit. 2021-04-03]. Dostupné z: https://github.com/google/filament.

46. *Sceneform* [online]. Google, [b.r.] [cit. 2021-04-03]. Dostupné z: https://developers.google.com/sceneform/develop.

47. *Google Sceneform – Is it deprecated? Any replacement?* [Online] [cit. 2021-04-03]. Dostupné z: https://stackoverflow.com/questions/62453399/google-sceneform-is-it-deprecated-any-replacement/62694454#62694454.

48. *Maintained Sceneform SDK for Android* [online]. Thomas Gorisse, [b.r.] [cit. 2021-04-03]. Dostupné z: https://github.com/ThomasGorisse/sceneform-android-sdk.

49. *Membership Details* [online]. Apple, [b.r.] [cit. 2021-04-01]. Dostupné z: https://developer.apple.com/programs/whats-included/.

50. *How the Program Works* [online]. Apple, [b.r.] [cit. 2021-04-01]. Dostupné z: https://developer.apple.com/programs/how-it-works/.

51. *Maxon* [online] [cit. 2021-04-19]. Dostupné z: https://www.maxon.net/en/.

52. *Olivia* [online] [cit. 2021-03-24]. Dostupné z: https://www.mixamo.com/.

53. *DVBSCENE* [online] [cit. 2021-04-19]. Dostupné z: https://dvb.org/wp-content/uploads/2019/12/dvb-scene44.pdf.

54. *Xbox Kinect and Infrared Camera* [online] [cit. 2021-03-28]. Dostupné z: https://www.youtube.com/watch?v=0Yr5_0imcbY.

55. POLCER, Michal. *Využití technologie Kinect při získávání motion capture dat.* 17. listopadu 2172/15. 708 00 Ostrava-Poruba, 2013. Magisterská práce. Vysoká škola Báňská.

56. *PHOTO: Face ID Infrared Dot Matrix* [online]. 2017 [cit. 2021-03-28]. Dostupné z: https://forums.macrumors.com/threads/photo-face-id-infrared-dot-matrix.2084242/.

57. *'Star Wars: The Force Awakens' Considered Making Snoke a Woman, Plus More Big Production Changes* [online]. 1 Manhattanville Rd, Suite 202, New York: Townsquare Media, Inc., 2015 [cit. 2021-03-28]. Dostupné z: https://screencrush.com/the-force-awakens-snoke-production-changes/.

58. *Studentská licence Cinema4D* [online] [cit. 2021-03-26]. Dostupné z: https://www.cinema4d.cz/education/edu-licensing/students-and-teachers/.

59. *Plans and Pricing* [online] [cit. 2021-04-01]. Dostupné z: https://www.maxon.net/en/buy.

60. *Blender is Free Software* [online] [cit. 2021-04-01]. Dostupné z: https://www.blender.org/about/license/.

61. *Unlock educational access to Autodesk products* [online] [cit. 2021-04-01]. Dostupné z: https://www.autodesk.com/education/edu-software/overview?sorting=featured&page=1.

62. *3D computer animation, modeling, simulation, and rendering software* [online] [cit. 2021-04-01]. Dostupné z: https://www.autodesk.com/products/maya/overview?term=1-YEAR.

63. *3D modeling and rendering software for design visualization, games, and animation* [online] [cit. 2021-04-01]. Dostupné z: https://www.autodesk.com/products/3ds-max/overview?term=1-YEAR.

64. *Unity Student* [online] [cit. 2021-04-01]. Dostupné z: https://store.unity.com/academic/unity-student.

65. *Choose the plan that is right for you* [online] [cit. 2021-04-01]. Dostupné z: https://store.unity.com/compare-plans.

66. *FREQUENTLY ASKED QUESTIONS (FAQ)* [online] [cit. 2021-04-01]. Dostupné z: https://www.unrealengine.com/en-US/faq.

67. *About* [online] [cit. 2021-04-19]. Dostupné z: https://www.blender.org/about/.

68. *Blender GitHub* [online] [cit. 2021-03-25]. Dostupné z: https://github.com/blender/blender.

69. *3D computer animation, modeling, simulation, and rendering software* [online] [cit. 2021-04-19]. Dostupné z: https://www.autodesk.com/products/maya/overview.

70. *3D modeling and rendering software for design visualization, games, and animation* [online] [cit. 2021-04-19]. Dostupné z: https://www.autodesk.com/products/3ds-max/overview.

71. *Mixamo* [online] [cit. 2021-03-25]. Dostupné z: https://helpx.adobe.com/creative-cloud/faq/mixamo-faq.html.

72. *Mixamo* [online] [cit. 2021-03-28]. Dostupné z: https://www.mixamo.com/.

73. *Houdini* [online] [cit. 2021-04-19]. Dostupné z: https://www.sidefx.com/products/houdini/.

74. *Houdini* [online] [cit. 2021-04-19]. Dostupné z: https://www.sidefx.com/products/houdini-apprentice/.

75. *Tips for optimizing tracking* [online]. Google, [b.r.] [cit. 2021-04-19]. Dostupné z: https://developers.google.com/ar/develop/java/augmented-images#tips_for_optimizing_tracking.

76. *Only one texture applied on model that consists of many textures 39* [online] [cit. 2021-04-21]. Dostupné z: https://github.com/ThomasGorisse/sceneform-android-sdk/issues/39.

77. *augmentedImages sample 42* [online] [cit. 2021-04-21]. Dostupné z: https://github.com/ThomasGorisse/sceneform-android-sdk/pull/42.

78. *Tips for selecting reference images* [online]. Google, [b.r.] [cit. 2021-04-21]. Dostupné z: https://developers.google.com/ar/develop/java/augmented-images#tips_for_creating_the_image_database.

79. *Tips for creating the image database* [online]. Google, [b.r.] [cit. 2021-04-21]. Dostupné z: https://developers.google.com/ar/develop/java/augmented-images#tips_for_selecting_reference_images.

80. *App Manifest Overview* [online]. Google, [b.r.] [cit. 2021-04-19]. Dostupné z: https://developer.android.com/guide/topics/manifest/manifest-intro.

81. CINEMA4DTUTORIAL.NET. *IRON MAN 3 MARK 42 FREE 3D MODEL* [online]. 2014 [cit. 2021-04-25]. Dostupné z: https://www.cinema4dtutorial.net/?p=2969.

82. RENDERPEOPLE. *3D Animated People* [online] [cit. 2021-04-25]. Dostupné z: https://renderpeople.com/free-3d-people/.

83. THEBIGGYSMITH. *Matrix Raining Code* [online]. 2013 [cit. 2021-04-25]. Dostupné z: https://www.youtube.com/watch?v=kqUR3KtWbTk.

84. MAXBUGOR. *Character Knight 3D Model* [online]. 2020 [cit. 2021-04-29]. Dostupné z: https://free3d.com/3d-model/character-knight-5664.html.

85. *GLTF EXPORT* [online] [cit. 2021-04-19]. Dostupné z: https://labs.maxon.net/?p=3360.

Příloha A

Zdrojový kód aplikace

V souborové příloze této práce můžete nalézt zdrojový kód aplikace, která vznikala s tímto textem a je na ni mnohokrát odkazováno. Dále pak potřebné instalační soubory aplikace, použité modely, AR tagy a video ukázky funkční aplikace. V tabulce **A.1 (Soubory přílohy)** je uvedena konkrétní struktura přílohy. Cesty jsou uvedeny relativně z adresáře *Priloha* ve Windows formátování (zpětné lomítka).

Tabulka A.1: Soubory přílohy

Cesta	**Obsah**
\apk\	Obsahuje instalační balíčky ARCore ve verzích pro emulátor
\build\	Obsahuje již sestavené instalační balíčky aplikace
\pouzite_modely\	Zde jsou v aplikaci použité modely v exportované variantě
\sceneform-android-sdk-SovinecAR\	Hlavní adresář projektu Sceneform SDK do Android Studia
\sceneform-android-sdk-SovinecAR\ maiwavo\sovinecar\	Hlavní adresář modulu do projektu Sceneform SDK, zde se nachází zdrojové kódy a soubory mé aplikace
\tagy\	Obsahuje spouštěcí tagy pro použití v aplikaci
\ukazky\	Obsahuje ukázkové videa aplikace natočené na reálných telefonech a emulátoru
\ARCore_for_emulator.html	Odkaz na stažení instalačních balíčků ARCore z GitHubu
\ARCore_supported.html	Odkaz na tabulku telefonů podporujících knihovnu ARCore
\Model_viewer.html	Odkaz, kde si lze přiložené modely online a zdarma prohlédnout
\GitHub_repo.html	Odkaz na můj GitHub repozitář s projektem *sceneform-android-sdk*
\readme.txt	Soubor s návodem a důležitými informacemi pro spuštění mé aplikace na telefonu nebo emulátoru